UNIVERSITÀ DI PISA

Corso di Laurea in Informatica Umanistica

Relazione

Facebook e il referendum costituzionale: una sentiment analysis

Candidato: *Gabriele Cocchi*

Relatore: *Prof. Alessandro Lenci*

Correlatore: *Prof. Felice Dell'Orletta*

Anno Accademico 2016-2017

INDICE

1. LA SENTIMENT ANALYSIS

Gli studi di *sentiment analysis* consistono nell'utilizzo di strumenti di *Natural Language Processing* e di linguistica computazionale per determinare il "tono emozionale", l'atteggiamento e la reazione emotiva di chi parla o scrive riguardo a uno specifico argomento. Nell'ultimo decennio si sono susseguite articolate ricerche su diverse forme di *sentiment analysis*, "from tools to analyse product reviews to more complex tasks such as understanding and predicting political voting from social media" (Maynard e Bontcheva, 2016, p. 1142). Le prime ricerche sulla *sentiment analysis* – non a caso conosciuta anche come *opinion mining* (estrazione delle opinioni) – risalgono ai primi anni 2000, con alcuni studi come quelli di Turney (Turney, 2002) o di Bo Pang e Lilian Lee (Pang e Lee, 2002). La massiccia diffusione sul Web dei social network, dei forum e dei blog ha reso disponibili una grandissima quantità di testi che, per gli studi di linguistica computazionale, possono funzionare da *corpora*, ma più specificatamente, nel caso delle ricerche di *sentiment analysis*, possono costituire validi terreni di prova per cercare di scattare una fotografia attendibile del tono emozionale di chi scrive riguardo a un determinato argomento. Da qui, ad esempio, la spiccata utilità della *sentiment analysis* in ambito di marketing (Pang e Lee, 2008, pp. 56-57): la possibilità, da un lato, di esprimere un proprio parere personale su social network, gruppi di discussione e siti commerciali e quella, dall'altro, di analizzare questa immensa mole di dati per stabilire le tendenze degli utenti nei confronti di un prodotto o di un servizio (oppure, in politica, l'accoglienza riservata a una riforma, ad esempio) rendono gli studi di *sentiment analysis* una sfida sempre più al passo coi tempi. Anche se di complessa applicazione informatica, come nel caso della valutazione dell'efficienza degli strumenti di *sentiment analysis*: "(…) fair evaluation of such systems is fraught with difficulty, partly because the task is often subjective (human annotators do not agree on what is correct), and partly because comparing systems fairly is

complicated when they tackle the problem in different ways" (Maynard e Bontcheva, 2016, p. 1142).

1.1. LE ORIGINI DELLA SENTIMENT ANALYSIS: UN BREVE CENNO STORICO

Pang e Lee quindici anni fa intuivano già le potenzialità, soprattutto nell'ambito del marketing (ma oggi anche in quello della politica), della *sentiment analysis*:

> "Today, very large amounts of information are available in on-line documents. As part of the effort to better organize this information for users, researchers have been actively investigating the problem of automatic text categorization. The bulk of such work has focused on *topical* categorization, attempting to sort documents according to their subject matter (…) However, recent years have seen rapid growth in on-line discussion groups and review sites (…) where a crucial characteristic of the posted articles is their *sentiment*, or overall opinion towards the subject matter – for example, whether a product review is positive or negative. Labeling these articles with their sentiment would provide succinct summaries to readers (…) Sentiment classification would also be helpful in business intelligence applications (…) and recommender systems (…) where user input and feedback could be quickly summarized; indeed, in general, free-form survey responses given in natural language format could be processed using sentiment categorization" (Pang e Lee, 2002, p. 79).

Utilizzando un insieme di 2053 recensioni cinematografiche online come corpus, attraverso metodi di *Machine Learning* applicati ai testi, Pang e Lee riuscirono a determinare se il giudizio espresso in una recensione fosse positivo o negativo, con un'accuratezza (la percentuale di testi classificati correttamente) che, a seconda degli algoritmi impiegati, oscillava tra il 78.7% e l'82.9% (2002). Con una nota a margine di non poco conto, i metodi di *Machine Learning* risultavano di gran lunga più efficaci rispetto ad analisi basate sulla conoscenza umana (ad esempio una lista di parole che funzionasse da indicatore di positività e negatività), la cui accuratezza era sensibilmente inferiore: tra il 58% e il 64% (2002, p. 81).

Il compito era – ed è – tanto impegnativo quanto stimolante: si tratta(va) di passare dalla consueta classificazione dei testi basata su argomenti (identificabili tramite parole chiave) a una più sottile – e complessa – categorizzazione incentrata sul *sentiment* espresso dallo scrivente nel testo, che non costituisce un semplice sottoinsieme della prima, "with the two "topics" being positive sentiment and negative sentiment" (Pang e Lee, 2002, p. 81). Gli algoritmi, di conseguenza, devono sviluppare una maggiore capacità di comprensione dei testi rispetto a quella necessaria nella consueta classificazione per argomento:

"A challenging aspect of this problem that seems to distinguish it from traditional topic-based classification is that while topics are often identifiable by keywords alone, sentiment can be expressed in a more subtle manner. For example, the sentence "How could anyone sit through this movie?" contains no single word that is obviously negative. (…) Thus, sentiment seems to require more *understanding* than the usual topic-based classification" (Pang e Lee, 2002, p. 79).

Anche se oggi, all'interno dell'insieme degli studi di *sentiment analysis*, si possono individuare altri due sottotipi, divisi da una differenza di fondo: da una parte una ricerca che mira a identificare la semplice polarità (positiva, negativa, neutra o mista nel caso di questo lavoro) del testo analizzato (*plain polarity sentiment analysis)*, dall'altra una s*entiment analysis* per argomenti (*topic based* o *aspect-based sentiment analysis*) (Bitext, *Differences between polarity and topic-based sentiment analysis*).

Qual è la differenza? La prima tiene conto dei termini positivi o negativi presenti in un determinato testo: la sua utilità è notevole quando si tratta di strutturare *dataset*:

"Let say I have 1000 reviews on my product that I want to analyze. By using polarity I can identify that 30% are negative, 20% are neutral and 50% are positive – and that's good for segmentation. But I am left with three chunks of 300, 200 and 500 reviews to go through if I want to get more meaningful insights, rather than just a nice looking pie chart" (Bitext).

Di seguito un esempio di come verrebbero trattate tre frasi differenti nel contesto di una *polarity analysis*:

1. "*The weather is amazing, and I love my XYZ phone*". La polarità in questo caso è positiva, data la presenza di termini come "*amazing*" e "*love*";
2. *"The weather is gloomy and sad, and I hate my XYZ phone".* Polarità negativa, data la presenza di termini come *"gloomy","sad"* e *"hate";*
3. "*The weather is gloomy and sad, and I love my XYZ phone*". In questo caso alcuni termini sono associati ad una polarità positiva, altri ad una polarità negativa (Bitext).

È proprio in quest'ultimo caso che si rende utile una *topic based analysis*: riuscire a identificare a cosa l'opinione dello scrivente si riferisce, infatti, significa che "the true expressed opinion is not lost in a misleading overall score" (Bitext). Nel caso delle frasi riportate sopra, ad esempio, l'azienda immaginaria che produce i cellulari "XYZ" sarebbe in grado di circoscrivere la sua analisi di marketing soltanto alle opinioni espressamente riferite ai cellulari "XYZ", tralasciando in questo modo il "rumore" di fondo, cioè, negli esempi sopra, i commenti sulle condizioni atmosferiche. Assume un'importanza fondamentale, quindi, il compito di individuare soltanto le espressioni che recano un *sentiment* collegato all'argomento oggetto di analisi. Per farlo "we need to rely on linguistic knowledge" (Bitext), facendo ricorso a metodi come il *POS tagging*, il *parsing* e gli n-grammi. Le conoscenze linguistiche infatti, svelando la struttura della frase analizzata, permettono di creare metodi di *Machine Learning* in grado di rilevare il *sentiment topic* a cui l'opinione espressa nel testo è riferita. Nel caso di queste due semplici frasi, ad esempio:

1. *"Adoro la riforma costituzionale di Renzi".* L'argomento a cui l'opinione espressa dal verbo "adorare" si riferisce sarà il complemento oggetto, cioè la "riforma costituzionale";

2. *"La riforma costituzionale di Renzi è una schifezza"*. La presenza del verbo "essere" indica che *il sentiment topic* in questo caso sarà da individuare nel soggetto della frase, "la riforma costituzionale".

Tornando per un'ultima volta alle origini della *sentiment* analysis, lavori precedenti alle ricerche di Turney, Pang e Lee, relativi a categorizzazione di testi *non topic-based*, si erano concentrati invece sulla classificazione dei documenti in base alla loro fonte, "with statistically detected variation (…) serving as an important cue", oppure in base al loro genere letterario (Pang e Lee, 2002). Altri tentativi sono stati dedicati alla ricerca di caratteristiche del testo che indicassero l'utilizzo di un "linguaggio soggettivo" ("subjective language"), senza specificare però in cosa quel "linguaggio soggettivo" consistesse:

> "But, while techniques for genre categorization and subjectivity detection can help us *recognize* documents that express an opinion, they do not address our specific *classification* task of determining what that opinion actually is" (Pang e Lee, 2002).

1.2. OBIETTIVI

Il presente lavoro, prendendo spunto dalle succitate ricerche in ambito di *sentiment analysis*, si è focalizzato sull'argomento del referendum costituzionale del 4 dicembre 2016 e, più in particolare, sui commenti registrati nelle pagine Facebook di otto tra i principali quotidiani nazionali (tutti anche cartacei, tranne uno: Huffingtonpost.it) prima dello svolgimento della consultazione elettorale.

Con un duplice obiettivo di fondo: da una parte determinare il *sentiment*, il tono emozionale in merito alla riforma costituzionale di chi ha commentato sulle pagine Facebook dei quotidiani, e dall'altra creare a tale scopo validi ed efficienti modelli di calcolo.

La ricerca è stata svolta su un corpus di post e commenti che logicamente fanno riferimento – per l'efficacia e l'attendibilità dello studio – a un intervallo di tempo precedente alla data di svolgimento del referendum, utilizzando il linguaggio di programmazione *Python.*

Il lavoro si è articolato in quattro fasi: la raccolta del corpus (v. 2), il filtraggio del corpus in modo da poter considerare soltanto post e commenti effettivamente attinenti all'argomento del referendum costituzionale (v. 3), la creazione di tre modelli di calcolo per determinare la polarità emotiva dei commenti (v. 4) e infine la valutazione delle performance predittive di tali modelli di calcolo (v. 5).

2. RACCOLTA DEL CORPUS

Il primo passo di questo lavoro è stato la creazione di un corpus popolato da post e commenti Facebook che facessero esplicito riferimento all'argomento della riforma costituzionale e/o del relativo referendum del 4 dicembre 2016, e che fossero risalenti a un medesimo intervallo di tempo, ovviamente antecedente (ai fini della coerenza della ricerca) alla data di svolgimento della consultazione elettorale. In questo senso la validità della ricerca non viene inficiata dal fatto che sia stata portata a termine alcuni mesi dopo il 4 dicembre 2016, e non prima: il fattore essenziale infatti è che il corpus sia costituito da post e commenti registrati sulle pagine Facebook dei quotidiani in un periodo di tempo precedente al 4 dicembre. Una volta messi da parte i dati raccolti, paradossalmente, la ricerca si sarebbe potuta effettuare anche due anni dopo il 4 dicembre 2016 (o persino più avanti) e ciò nonostante la sua fondatezza non sarebbe stata compromessa.

Segnatamente l'intervallo di tempo scelto, all'interno del quale rientrano tutti i post e i relativi commenti raccolti, è di circa sei mesi e, nello specifico, va dal 1 maggio 2016 al 16 novembre 2016, immediatamente a ridosso della consultazione elettorale. I quotidiani nazionali online da cui si è attinto sono otto: Il Corriere della Sera, Il Fatto Quotidiano, Il Giornale, Huffingtonpost.it, Il Messaggero, La Repubblica, il Sole 24 Ore e la Stampa. In una prima fase, per la raccolta dei post e dei commenti dalle pagine Facebook dei quotidiani elencati sopra, è stato utilizzato un *crawler* già predisposto appositamente per questo *task*, fornito dal CoLing Lab del Dipartimento di Filologia, Letteratura e Linguistica dell'Università di Pisa, a cui sono state apportate piccole modifiche, coerenti con gli obiettivi di questo lavoro. Il *crawler* in questione, non a caso, viene definito *"backwards crawler*", letteralmente "*crawler* all'indietro", per la sua caratteristica di raccogliere post e commenti a partire dalla data che chiude l'intervallo di tempo indicato, per poi procedere, appunto, all'indietro fino al raggiungimento della data che apre l'intervallo.

Questa fase, però, non ha esaurito il compito di definizione di un corpus che fosse adatto alle esigenze di questa ricerca, poiché i post e commenti scaricati dal "*backwards crawler*" non fanno riferimento a un argomento specifico (in questo caso il referendum costituzionale), ma a quelli più disparati, pubblicati ogni giorno a ogni ora dai quotidiani selezionati nel corso dei sei mesi presi in considerazione. È per questo che si è resa necessaria una seconda fase di filtraggio dei dati raccolti con il *crawler* (v. 3), in modo da poter considerare esclusivamente post e commenti effettivamente attinenti all'argomento di questo lavoro, scartando invece tutti gli altri. Di seguito il codice *Python* che è stato necessario modificare per adattare il *crawler* alle esigenze di questa ricerca:

```
#Token to access facebook graph
access_token = 'EAACEdEose0cBAOQDUkiHIGW8EK6hW2m7fYI2/
qp7mTiK9Jj7nfF9XE32fnl8GpQDOZApIqggQPrUYwDyS9ARVuwxEXn
IfkTVJvQhXbu34JC2fE0AAZCgGw2EBxxZA39h8lfySmYXR0Jpog9HC
tvCVIMnWize1GES8xG3GhpHowZDZDF
access_token_2 = ''

inittimestamp = "1462053600"

#contatori per post e commenti
counter_posts = 0
counter_comments = 0

#aggiornamento e ultima timestamp
update = True
current_timestamp = 0

user_list = {}
current_user = ""
oldest_timestamp = 0
```

```
proxies = {"http": "htpp://131.114.56.15:3128",
"https": "http//131.114.56.15:3128",}
```

Access_token è una variabile a cui è stato necessario assegnare, durante ogni sessione di lavoro, il token d'accesso per leggere i dati relativi alle pagine Facebook dei quotidiani. Un token d'accesso non è altro che "una stringa opaca che identifica un utente, un'app o una Pagina e può essere usata dall'app per le chiamate API Graph (…) Inoltre, i token includono informazioni sulla loro scadenza e sull'app che li ha generati" (Facebook for developers).

Inittimestamp è un'altra variabile, che rappresenta la data a cui il *crawler* deve arrivare, procedendo all'indietro, nello scaricamento dei post e commenti dalle pagine selezionate. In particolare, a *inittimestamp* viene assegnato un valore di tempo Unix, detto anche "*Unix timestamp*".

In un semplice file .csv, invece, è contenuta la lista degli utenti (cioè delle pagine Facebook dei quotidiani) da cui scaricare post e commenti, con associata la *Unix timestamp* più vecchia, a cui il crawler è arrivato nel suo processo di "scaricamento all'indietro".

	A	B
1	HuffPostItalia,1464070304	

Figura1. Esempio di file .csv contenente il nome della pagina dell'Huffingtonpost.it con associata una *Unix timestamp* corrispondente a martedì 24 maggio 2016

In questo lavoro si è preferito inserire un utente alla volta, per poi passare al successivo, in modo da avere la certezza che nel processo, per ogni singolo quotidiano, venissero scaricati correttamente tutti i dati desiderati. Finita la lettura dei dati per un determinato quotidiano *x*, il file .csv viene modificato, inserendo il nome utente della pagina Facebook del quotidiano *y*, associato

alla data (in formato *Unix timestamp*) da cui (ri)partire nel processo di *crawling*. Avviando il *crawler* da terminale, come un normale script in *Python*, al termine del processo si sono ottenuti 3,4 gigabyte di post e relativi commenti. Di seguito un esempio tratto da uno dei file di *output* ottenuti, con un post del Corriere della Sera e tre commenti figli:

```
<doc user="corrieredellasera"
id="284515247529_10154415439297530" type="post"
parent_post="" parent_comment=""  date="1477509781"
location="" likes="301" comments="12" shares="63">

Un utente accende la telecamera subito a ridosso della
nuova scossa sismica, esattamente alle 19.13
[Terremoto 5.6 Marche: la scossa in una casa ad
Ancona]

</doc>

<doc user="corrieredellasera"
id="10154415439297530_10154415976772530"
type="comment"
parent_post="284515247529_10154415439297530"
parent_comment="" date="1477514393" likes="5"
comments="3">

il terremoto che arriverà dalle urne il 4 dicembre
seppellirà tutti i politici

</doc>

<doc user="corrieredellasera"
id="10154415439297530_10154416138217530"
type="comment"
```

```
parent_post="284515247529_10154415439297530"
parent_comment="10154415439297530_10154415976772530"
date="1477519895" likes="0" comments="0">

Hai proprio il senso dell'opportunità....ridicolo

</doc>

<doc user="corrieredellasera"
id="10154415439297530_10154416473632530"
type="comment"
parent_post="284515247529_10154415439297530"
parent_comment="10154415439297530_10154415976772530"
date="1477531611" likes="0" comments="0">

Speriamo...

</doc>
```

Il passo successivo è stato quello di sottoporre il testo dei file ottenuti a *Part-of-speech tagging (POS tagging),* cioè il processo di identificazione, appunto, della parte del discorso di ogni *token*. In questo senso un *POS tagger* "is a piece of software that reads text in some language and assigns parts of speech to each word (and other token), such as noun, verb, adjective, etc." (Nlp.Stanford, *Stanford Log-Linear Part-Of-Speech Tagger*). Processo effettuato, ancora, con i *tools* del CoLing Lab. Di seguito un esempio di struttura di uno dei file in seguito al *POS tagging:*

```
<doc user="corrieredellasera"
id="284515247529_10154446006852530" type="post"
parent_post="" parent_comment="" date="1478260201"
location="" likes="45" comments="36" shares="6">

1     4     4     N     N     _
```

```
2    Novembre   Novembre   S    SP    _
3    ,    ,     F    FF    _
4    la   il    R    RD    num=s|gen=f
5    gelida     gelido     A    A     num=s|gen=f
6    stretta    stretta    S    S     num=s|gen=f
7    di   di    E    E     _
8    mano mano  S    S     num=s|gen=f
9    Renzi-Raggi     Renzi-Raggi     N    N     _
10   :    :     F    FC    _
11   «    «     F    FB    _
12   Ciao ciao  I    I     _
13   Sindaco    Sindaco    S    SP    _
14   »    »     F    FB    _
15   :    :     F    FC    _
16   http://bit.ly/2flRZ62 http://bit.ly/2flRZ62 N    N
     _
17   [    [     F    FB    _
18   Timeline   Timeline   S    SP    _
19   Photos     Photos     S    SP    _
20   ]    ]     F    FB    _

</doc>

<doc user="corrieredellasera"
id="10154446006852530_10154446186572530"
type="comment"
parent_post="284515247529_10154446006852530"
parent_comment="" date="1478261246"
likes="27" comments="3">

1    Occhio     occhio     S    S     num=s|gen=m
2    sindaca    sindacare  V    V
     num=s|per=3|mod=i|ten=p
3    ,    ,     F    FF    _
4    .... ....  N    N     _
5    sopra sopra B    B     _
6    Roma  Roma  S    SP    _
7    c'   ci    P    PC    num=n|gen=n
8    è    essere     V    V
     num=s|per=3|mod=i|ten=p
9    la   il    R    RD    num=s|gen=f
10   parata     parata     S    S     num=s|gen=f
11   di   di    E    E     _
12   scie scia  S    S     num=p|gen=f
13   chimiche11 chimiche11 S    S     num=n|gen=n
```

Come già detto, in una seconda fase di filtraggio, è stato necessario "ripulire" i file di output prodotti attraverso il *crawling*, ottenendo, sì, un corpus più ridotto, ma contenente post e commenti esclusivamente riguardanti il referendum costituzionale.

3. FILTRAGGIO DEL CORPUS

Per il filtraggio dei file ottenuti nella prima fase del lavoro è stato necessario creare da zero un apposito *script* in *Python* (v. 9.1). Inizialmente, sono stati selezionati solamente i file *postaggati* (terminanti in ".pos") e per ognuno di essi, in via provvisoria, si è stabilita una variabile *preleva_post* con valore booleano uguale a *False*. Come a dire: se non vengono soddisfatte le condizioni successive – e quindi la variabile *preleva_post* mantiene sempre lo stesso valore booleano – allora il post e i relativi commenti figli appena analizzati non devono essere prelevati e copiati in un file temporaneo ("filebuffer.txt").

```
for file in os.listdir(cwd):
   if file.endswith(".pos"):
      in_file = open(file, "r")
      preleva_post = False
      ribuffer = open("filebuffer.txt", "wb")
      words = []
```

In seguito, è stato analizzato riga per riga ognuno dei file, verificando tre diverse possibili condizioni:

1. La riga che si sta prendendo in considerazione rappresenta l'inizio di un post, se inizia con la stringa '<doc' e la stringa 'type="post"' è presente nella riga stessa (v. 2).

```
if line.startswith("<doc") and 'type="post"' in
line:
    type = "post"
    if preleva_post == True:
        print "trovato"
        ribuffer.close()
```

```
            lavora_ribuffer("filebuffer.txt",
"Output.txt")
            preleva_post = False
            ribuffer = open("filebuffer.txt", "wb")
            words = []
```

In tal caso alla variabile *type*, come valore, viene assegnata la stringa 'post' e, soltanto se alla variabile *preleva_post* è associato un valore booleano uguale a *True*, viene invocata la funzione "lavora_ribuffer" – definita in precedenza, con argomenti il file temporaneo citato prima e il file di output globale ("Output.txt") – che invoca a sua volta la funzione "FileSave".

```
def lavora_ribuffer(namebuffer,filenameout):
   with open(namebuffer, "r") as content:
      true_content = content.read()
   FileSave(filenameout,true_content)

def FileSave(filename,content):
   with open(filename, "a") as myfile:
      myfile.write(content)
```

In altre parole, il contenuto del file temporaneo viene copiato nel file di output globale, alla variabile *preleva_post* è assegnato nuovamente il valore booleano di *False*, il file temporaneo viene riaperto in modalità "write" e in questo modo azzerato, e infine anche la lista *words* (di cui si vedrà in seguito l'utilità) viene azzerata.

Nel caso in cui, invece, la variabile *type* non abbia associato il valore booleano di *True* (cioè, quindi, il post analizzato non faccia riferimento al referendum costituzionale), il file temporaneo viene riaperto in modalità "write" e, anche qui, azzerato.

2. La riga che si sta prendendo in considerazione rappresenta la fine di un post (e quindi l'inizio di un commento), se inizia con la stringa '</doc>' e la variabile *type* ha associato, come valore, la stringa 'post'.

```
if line.startswith("</doc>") and type == "post":
    type = "comment"
    for i,j in zip(words, words[1:]):
        if ((i == "referendum" or i == "Referendum") and j == "costituzionale") or ((i == "riforma" or i == "Riforma") and (j == "costituzionale" or j == "Boschi")) or ((i=="ddl" or i == "Ddl") and (j=="Boschi" or j == "costituzionale")) or (i == "4" and j == "dicembre") or (i == "quesito" and j == "referendario"):
            preleva_post = True
```

In tal caso si procede ad analizzare la lista *words*, all'interno della quale vengono inseriti, per ogni post, tutti i relativi *token* che compongono il titolo del post stesso. Viene utilizzata la *built-in function* di *Python zip*, che produce una lista di tuple, dove "the *i*-th tuple contains the *i*-th element from each of the argument sequences or iterables" (Docs.Python, *Built-in Functions*). *Zip* viene utilizzata sulla lista *words* e sulla lista *words* eccetto il primo elemento, di modo che la lista di tuple avrà la seguente forma, con, ad esempio, *words* uguale a "['Ecco', 'la', 'riforma', 'costituzionale', 'di', 'Renzi']":

```
[('Ecco', 'la'), ('la', 'riforma'), ('riforma', 'costituzionale'), ('costituzionale', 'di'), ('di', 'Renzi')]
```

In questo modo, considerando di fatto i bigrammi del titolo di ogni post, è possibile verificare se siano presenti un set di espressioni chiave ("referendum costituzionale", "riforma costituzionale", "riforma Boschi", "ddl Boschi", "ddl costituzionale, "4 dicembre", "quesito referendario") e capire quindi se il post faccia riferimento all'argomento del referendum, e quindi vada preso in considerazione, oppure no.

È stato necessario infatti circoscrivere il campo della ricerca, utilizzando espressioni chiave più specifiche, visto che, ad esempio, nel caso si fosse utilizzata la parola chiave "referendum" per il filtraggio (e non "referendum costituzionale"), lo *script* avrebbe considerato anche post attinenti ad argomenti non pertinenti, come il referendum della Brexit, quello italiano sulle trivelle o quello in Bulgaria del novembre 2016. Con espressioni più specifiche, invece, si ottiene un numero di risultati certamente minore, con la certezza, però, che l'argomento dei post selezionati non cada al di fuori del campo della ricerca.

Se anche una sola delle parole chiave riportate sopra è presente nella lista *words*, cioè nel titolo del post, allora alla variabile *preleva_post* viene assegnato il valore booleano di *True*.

3. La riga che si sta prendendo in considerazione non è né l'inizio né la fine di un post, ma una delle righe del titolo tra l'inizio e la fine, a patto che alla variabile *type* sia associata la stringa 'post'.

```
elif type == "post":
    sequenza_parole = line.split( )
    if len(sequenza_parole) != 0:
        parola_colonna_3 = sequenza_parole[2]
        words.append(parola_colonna_3)
```

In tal caso la riga di testo viene divisa utilizzando il metodo *split()*, che restituisce una lista contenente tutte le parole che la compongono (il separatore in questo caso è lo spazio bianco). Nella lista *words*, con il metodo *append()*, viene inserita nello specifico la parola che figura nella terza colonna del file *postaggato*, per la successiva analisi tramite espressioni chiave all'interno della condizione 2.

Il comando chiave, al di fuori delle tre condizioni spiegate sopra, è quello che utilizza il metodo *write()* per scrivere ogni riga del testo all'interno del file temporaneo:

```
ribuffer.write(line)
```

Ricapitolando passo passo il funzionamento dello *script* di filtraggio del corpus: la prima riga di ogni file che la macchina incontrerà sarà l'inizio di un post, ma la variabile *preleva_post* sarà uguale a *False* (condizione 1); per le righe successive (quelle corrispondenti al titolo del post) verrà soddisfatta la condizione 3, e ogni *token* verrà inserito nella lista "word"; arrivati al termine del post (condizione 2), la macchina controllerà se all'interno di *words* è presente almeno una delle espressioni chiave ricercate e, in caso affermativo, a *preleva_post* verrà assegnato il valore di *True*; in questo modo – tenendo a mente che ogni riga del testo viene scritta nel file temporaneo – all'inizio del post successivo il file temporaneo (che in quel momento conterrà il post precedente e tutti i suoi commenti figli) verrà copiato nel file di output globale e poi azzerato, insieme a *words*, mentre *preleva_post* verrà settato nuovamente a *False*. In questo modo il ciclo si ripeterà fino alla fine del testo e il file di *output* risultante sarà composto esclusivamente da post e commenti in cui è presente almeno una delle espressioni chiave.

Il corpus ottenuto applicando lo *script* di filtraggio appena descritto è costituito da 96 447 commenti e 2 269 073 *token.*

4. CREAZIONE DEI MODELLI PER LA SENTIMENT ANALYSIS

Terminata l'operazione di filtraggio e ottenuto il corpus desiderato, il passo successivo è stato la creazione di tre distinti modelli per il calcolo della polarità emotiva di ognuno dei 96 447 commenti componenti il corpus. Come base di partenza sono stati utilizzati tre lessici, forniti dal CoLing Lab: ANEW.cfg2.bootstrapped.FB-NEWS15.ppmi.cos, ItEM.sorted.polarity.txt e ANEW.cfg2.bootstrapped.repubblica.itwac.ppmi.cos. Ogni lessico ha una semplice struttura, come nell'esempio seguente:

```
positive   luogotenenziale-a     0.003999
positive   interdittivo-a  0.003909
positive   idrico-a   0.003822
positive   refluo-a   0.003619
positive   trasfusionale-a 0.003364
positive   organizzativo-a 0.002696
positive   perinatale-a    0.002389
positive   clinico-a  0.002284
negative   egoista-a  0.586033
negative   incapace-a 0.579688
negative   insensibile-a   0.577093
negative   depresso-a 0.573844
negative   odioso-a   0.562596
negative   cattivo-a  0.561038
negative   infelice-a 0.559036
negative   vizioso-a  0.555424
```

Ad ogni lemma presente in un dato lessico *x* sono associati la relativa parte del discorso (il *tag* "-a" sta ad indicare un aggettivo, ad esempio), il suo valore di positività e quello di negatività (indicati rispettivamente dalla presenza nella riga di testo di *"positive"* e *"negative").* Nel lessico

ItEM.sorted.polarity.txt, a differenza degli altri due, i dati sono già stati filtrati e normalizzati per *z-score*. "A z-score, in simple terms, is a score that expresses the value of a distribution in standard deviation with respect to the mean" (Madhavan, 2015):

$$z = \frac{x - \mu}{\sigma}$$

Dove x è un valore in una distribuzione, μ è la media della distribuzione e σ rappresenta la deviazione standard (o scarto quadratico medio). Lo *zscore*, in altre parole, rappresenta la distanza, in termini di deviazione standard, di un valore x dalla media della popolazione di dati considerati. È infatti dimostrato come l'utilizzo della *z-score* in qualità di *feature* nella *sentiment classification* migliori, in termini di efficienza, il risultato finale:

> "The evaluation demonstrates that Z_scores features can significantly improve the prediction performance (…) We suggest using a new type of features for Sentiment Analysis, Z_score can distinguish the importance of each term in each class" (Hamdan e altri, 2014, pp. 636-637).

ItEM (an Italian emotive Lexicon), più nello specifico, è un lessico emotivo appositamente costruito per la lingua italiana (Passaro e Lenci, 2016), nel quale ogni termine è collegato ad uno specifico *score* che stima la sua associazione con ognuna delle otto emozioni della classificazione di Plutchik (1994): GIOIA, TRISTEZZA, RABBIA, PAURA, FIDUCIA, DISGUSTO, SORPRESA E ASPETTATIVA. Uno dei maggiori punti deboli delle ricerche di *emotion detection*, infatti, è la scarsità di "risorse emotive", e "this problem is even more pressing for Italian" (Passaro e Lenci, 2016).

Partendo dal presupposto che la connotazione emotiva di una certa parola è il risultato sia della sua associazione sintagmatica che della sua associazione paradigmatica con le otto emozioni di cui sopra, ItEM è stato costruito "in a three stage process" (Passaro e Lenci, 2016). In una prima fase, anche con

l'aiuto di 60 parlanti madrelingua italiani, è stato costruito un piccolo lessico di "[347] emotive lemmas, highly associated to one or more Plutchik's basic emotions" (Passaro e Lenci, 2016). Per espandere questo lessico di base, in una seconda fase, sono stati estratti dal corpus La Repubblica (Baroni e altri, 2004) e itWac (Baroni e altri, 2009) "the list of the 30,000 most frequent nouns, verbs and adjectives, which were used as targets and contexts in a co-occurrence matrix collected using a five-word window centered on the target lemma" e, attraverso la costruzione di un *vector space model* per ogni coppia (emozione, PartofSpeech), sono stati calcolati gli *score* emotivi dei termini, come misura della "cosine similarity between the target lemmas and the centroid vectors" (Passaro e Lenci, 2016). Infine, ricorrendo nuovamente ad una annotazione umana (una classificazione su una scala da 1 a 5 dell'associazione di ogni parola con ciascuna delle otto emozioni), sono stati selezionati "the top 10 distinctive nouns, adjectives and verbs for each (emotion, PoS) pair" (Passaro e Lenci, 2016), al fine di espandere ulteriormente il lessico di base.

Tornando alla presente ricerca, l'obiettivo di fondo era quello di capire attraverso l'utilizzo di quale dei tre lessici (combinato con uno dei tre modelli di calcolo) si ottenesse il risultato migliore, in termini di aggregazione dei valori emotivi per ognuno dei commenti contenuti nel corpus. A questo scopo sono stati creati tre *script* in *Python*, ognuno da applicare al corpus di commenti utilizzando come riferimento ciascuno dei tre lessici (tranne ItEM.sorted.polarity.txt, già normalizzato, a cui sono stati applicati soltanto i primi due modelli). Per la creazione dei tre *script* sono state scelte tre differenti formule di calcolo della polarità (*pol*) di ogni commento (*C)*, come indicato di seguito (dove *m* rappresenta l'indice della prima parola di un dato commento, *n* quello dell'ultima, *pos* il valore di positività della parola C_i e *neg* quello di negatività):

1.

$$pol(C) = \sum_{i=m}^{n} pos(C_i) - neg(C_i)$$

2. $$pol(C) = \overline{pos} - \overline{neg}$$

dove:

$$\overline{pos} = \frac{1}{n}\sum_{i=m}^{n} pos(C_i)$$

e

$$\overline{neg} = \frac{1}{n}\sum_{i=m}^{n} neg(C_i)$$

3.

$$pol(C) = \sum_{i=m}^{n} posnorm(C_i) - negnorm(C_i)$$

dove *posnorm(C_i)* e *negnorm(C_i)* sono i valori di positività e negatività della parola *Ci* normalizzati, in modo che la loro somma, cioè, sia riportata a 1:

$$posnorm(C_i) = \frac{pos(C_i)}{pos(C_i) + neg(C_i)}$$

$$negnorm(C_i) = \frac{neg(C_i)}{pos(C_i) + neg(C_i)}$$

In prima battuta sono stati creati tre *script* corretti dal punto di vista dell'algoritmo in sé, ma non ottimizzati. Venivano utilizzati infatti due cicli (*for*), prima all'interno del corpus, in modo prendere in considerazione ciascuna riga di testo, e poi all'interno del lessico preso come riferimento. Ma gli *script* erano dispersivi in termini di tempo, ragion per cui si è optato per l'implementazione di una struttura che utilizzasse dei dizionari, facendo ricorso al modulo *collections.* Ad ogni chiave del dizionario (una determinata parola *x* presente nel lessico), in questo modo, sono stati associati due valori: quello di positività e quello di negatività della parola stessa, presenti nel lessico scelto. Evitando così, ad ogni riga di testo del corpus, di effettuare un ciclo su tutto il lessico preso in considerazione, ma eseguendo al contrario un più economico accesso per chiavi al dizionario. Di seguito alcuni stralci dello *script* che implementa la formula più semplice, quella al caso 1 di p. 22 (gli altri due script vengono riportati in appendice, v. 9.2, 9.3 e 9.4):

```
from collections import defaultdict
lexicon_dict = defaultdict(lambda:[0,0])
def CaricaLessico(lessico):
   with open(lessico) as lexicon:
      for line in lexicon:
         sequenza = line.split( )
         parola = sequenza[1]
         valore = sequenza[2].replace(',','.')
         if "positive" in line:
            lexicon_dict[parola][0] = valore
         elif "negative" in line:
            lexicon_dict[parola][1] = valore
```

In prima istanza viene definito il dizionario *lexicon_dict* - utilizzando la sottoclasse *defaultdict* del modulo *collections* - a cui, per ogni chiave, vengono assegnati due valori (*[0,0]*). In questo momento il dizionario è vuoto. Viene poi definita la funzione *CaricaLessico* che, prendendo in input un determinato lessico, lo apre, lo scorre riga per riga, divide ogni riga attraverso il metodo *split()* e inserisce le due stringhe che rappresentano i valori positivo e negativo associati a una determinata parola all'interno del dizionario *lexicon_dict*. In seguito la funzione viene invocata sul lessico scelto:

```
CaricaLessico("Scrivania/Lessici/ANEW.cfg2.bootstrappe
d.FB-NEWS15.ppmi.cos")
```

In questo momento il dizionario è costruito ed è formato da chiavi (le parole presenti nel lessico preso in considerazione) e da valori assegnati ad ognuna di esse (valore di positività e negatività della parola).

Il passo successivo consiste nell'analizzare il corpus popolato da post e commenti costruito in precedenza (v. 2).

```
in_file = open("Scrivania/Cocchi/Output.txt", "r")
tipo = "post"
contatore = 0
valore_emotivo = 0
pol_pos = 0
pol_neg = 0
ID = 'id="1533164030030322_1533359486677443"'
```

Viene definita una variabile *tipo* a cui viene assegnata come valore la stringa *"post"*, poiché il corpus inizia, appunto, con il primo dei post filtrati. Poi altre tre variabili (*valore_emotivo*, *pol_pos*, *pol_neg*), che rappresentano rispettivamente il valore aggregato, il valore di positività e quello di

negatività che risulteranno, in questo caso, dalla formula al caso 1 di p. 22. La variabile *ID* rappresenta invece l'identificatore del primo commento presente nel corpus.

Successivamente si procede a scorrere il corpus riga per riga, verificando, analogamente a quanto fatto nello *script* di filtraggio (v. 3), quattro diverse possibili condizioni, di cui le due più importanti sono le seguenti:

1. La riga che si sta prendendo in considerazione rappresenta l'inizio di un commento, se inizia con la stringa '<doc' e la stringa 'type="comment"' è presente nella riga stessa (per la struttura del corpus v. 2).

```
if line.startswith("<doc") and 'type="comment"'
in line:
    sequence = line.split( )
    ID_now = sequence[2]
    tipo = "comment"
    print "COMMENT"
```

   In tal caso la riga viene divisa, utilizzando come separatore lo spazio bianco, e il terzo elemento della lista che ne risulta (l'identificatore del commento) viene momentaneamente salvato nella variabile *ID_now*. Poi, soltanto se la variabile *contatore* è maggiore di zero, si passa allo *step* successivo.

```
if contatore > 0:
    with
open("Scrivania/Lessici/Emotional_Sum_wordbyword_
FB.txt", "a") as output:
        output.write(ID + ": " + "pol_pos:" +
repr(pol_pos) + " " + "pol_neg:" + repr(pol_neg)
+ " " + "aggr:" + repr(valore_emotivo) + "\n")
        ID = ID_now
```

```
            print contatore
            valore_emotivo = 0
            pol_pos = 0
            pol_neg = 0
```

Questo perché i valori di ogni commento vengono salvati nel file di *output* solamente all'inizio del commento successivo e da principio sono settati a zero: prenderli in considerazione subito, al primo commento del corpus, significherebbe scriverli inutilmente come uguali a zero nel file di *output*. Da qui l'utilità (limitata soltanto al primo commento) della variabile *contatore* (di aiuto anche per tenere traccia da terminale del numero di commenti analizzati).

Il valore aggregato, quello di positività e quello di negatività vengono scritti nel file di output e le tre rispettive variabili riportate a zero, in vista dell'analisi del commento successivo.

2. La riga che si sta prendendo in considerazione è una di quelle intermedie, all'interno di un commento, se la variabile *tipo* è uguale alla stringa "comment":

```
elif tipo == "comment":
    if line[0].isdigit():
        sequenza = line.split( )
        parola = sequenza[2]
        PoS = sequenza[3].lower()
        true_parola = parola + "-" + PoS
        positive =
float(lexicon_dict[true_parola][0])
        negative =
float(lexicon_dict[true_parola][1])
        valore_emotivo = valore_emotivo +
positive - negative
        pol_pos = pol_pos + positive
        pol_neg = pol_neg + negative
```

In tal caso, soltanto se il primo elemento della riga è costituito solamente da cifre (escludendo così sia le righe vuote che quelle che aprono e chiudono i commenti, non contenenti informazioni utili, v. 2), il lemma viene affiancato in un'unica stringa alla relativa parte del discorso, mentre alla variabile *valore_emotivo* viene sommato il corrispondente valore di positività e sottratto quello di negatività. La necessità di considerare anche la parte del discorso del lemma è facilmente intuibile: la parola "sapere", ad esempio, nel lessico è presente quattro volte, due (valore di positività e di negatività) con valore di verbo (con "v" come *tag*) e due con valore di sostantivo (con "s" come *tag*), ragion per cui si rende necessaria una distinzione tra i due casi.

Il file di *output* risultante ha questa struttura: ogni riga è composta, nell'ordine, dall'identificatore di ogni commento presente nel corpus, il valore di positività, quello di negatività e quello aggregato:

```
id="1533164030030322_1533359486677443": pol_pos:11.937171 pol_neg:11.918859 aggr:0.018311999999999995
id="1533164030030322_1533708583309200": pol_pos:11.992709999999999 pol_neg:11.558902 aggr:0.43380799999999975
id="1533164030030322_1533775349969190": pol_pos:3.114249 pol_neg:3.008029 aggr:0.10621999999999987
id="1533164030030322_1533264630020262": pol_pos:6.941181 pol_neg:6.493735999999999 aggr:0.4474449999999999
```

Figura 2. Esempio di file di output

La struttura dei due restanti *script* non differisce da quella appena presentata (v. 9.3 e 9.4), se non per l'operazione di calcolo dei valori.

5. VALUTAZIONE DELL'EFFICIENZA DEI MODELLI DI SENTIMENT ANALYSIS

Creati i modelli di calcolo del sentiment e ottenuti i relativi risultati, si è resa necessaria una valutazione della loro prestazione, in termini di performance di predizione della polarità di ogni commento. A questo scopo è stato creato un *gold standard*, un insieme (in una prima fase) di 100 commenti, scelti casualmente all'interno del corpus e annotati manualmente, che costituisse un valido riferimento per testare l'efficienza dei modelli di calcolo (si è scelto di annotarne manualmente soltanto 100 su 96 447 per ragioni di tempo). I 100 commenti sono stati annotati in un semplice file di testo secondo questa notazione:

```
id="1461396310541277_1005434672887848":  01
id="1424763700871205_1424800487534193":  01
id="1537150129632561_1537421186272122":  00
id="10154179956375958_10154180112015958":  01
id="10154720132141151_1831399290477547":  00
id="10154206013596151_10154206408146151":  01
id="10154429641327530_10154430630467530":  00
id="1528392483841659_1529933777020863":  01
id="10154488736667459_10154489479592459":  01
id="1513086642038910_1513197738694467":  01
id="1533164030030322_1535133223166736":  01
```

All'identificatore di ciascun commento (v. 2) è stata assegnata una categoria binaria, secondo questo criterio:

- 00 = commento con polarità neutra
- 10 = commento con polarità positiva
- 01 = commento con polarità negativa
- 11 = commento con polarità mista

Il fatto che la maggioranza dei commenti annotati manualmente all'interno del *gold standard* abbiano associata una polarità negativa non è un elemento così singolare, considerato che l'argomento preso in considerazione – il referendum costituzionale – ha comportato una netta divisione dell'opinione pubblica in schieramenti ben distinti.

Per valutare quanto i risultati ottenuti con i tre modelli di calcolo (v. 4) fossero coerenti con quelli annotati manualmente nel *gold standard*, è stato creato un apposito *script* in *Python* (v. 9.5).
Ogni coppia identificatore-polarità è stata inserita in un dizionario chiave-valore:

```
dict_gold = defaultdict(lambda:[0])
with open("Scrivania/Gold.txt", "r") as gold:
    for line in gold:
        sequenza = line.split()
        id_comment = sequenza[0]
        true_id = id_comment[:-1]
        dict_gold[true_id] = sequenza[1]
```

In un ulteriore dizionario, invece, sono state inserite le coppie identificatore-positività/negatività ottenute con i calcolatori, a patto però che i commenti considerati fossero gli stessi 100 presenti nel *gold standard*; vengono calcolati anche i valori massimi e minimi per positività e negatività:

```
dict_document = defaultdict(lambda:[0,0])
with
open("Scrivania/Lessici/Emotional_Sum_normalized_repub
blica.txt", "r") as documento:
    for line in documento:
        sequence = line.split()
        pos = float(sequence[1][8:])
        neg = float(sequence[2][8:])
       vero_id = sequence[0][:-1]
```

```
        if vero_id in dict_gold:
            dict_document[vero_id] = [pos, neg]
        if pos > massimo_pos:
            massimo_pos = pos
        elif pos < minimo_pos:
            minimo_pos = pos
        elif neg > massimo_neg:
            massimo_neg = neg
        elif neg < minimo_neg:
            minimo_neg = neg
```

Il passo successivo era trasformare i valori numerici dei file di *output* dei modelli di calcolo nella stessa notazione del file di *gold standard*, stabilendo due soglie (una per il valore di positività dei commenti, l'altra per quello di negatività), in base alle quali i valori numerici venissero trasformati nella notazione "00, 10, 01, 11". Ma come capire attraverso quali soglie si ottengono le performance di predizione migliori? A questo scopo sono state sperimentate più di 160 000 combinazioni, utilizzando come intervallo quello compreso tra i valori massimi e minimi di positività e negatività:

```
soglie_pos = np.linspace(minimo_pos, massimo_pos, 401)
soglie_neg = np.linspace(minimo_neg, massimo_neg, 401)
combinations = list(itertools.product(soglie_pos,
soglie_neg))
for element in combinations:
    for key in dict_document:
        pol_pos = float(dict_document[key][0])
        pol_neg = float(dict_document[key][1])
        SalvaPol(element[0], element[1], pol_pos,
pol_neg, key)
```

Per ognuna delle possibili combinazioni di soglie viene invocata la funzione *SalvaPol*, definita in precedenza, come il dizionario *dict_pred*:

```
def SalvaPol(pos_threshold, neg_threshold, pos_value,
neg_value, iD):
    if pos_value < pos_threshold and neg_value <
neg_threshold:
        dict_pred[iD] = "00"
    elif pos_value >= pos_threshold and neg_value <
neg_threshold:
        dict_pred[iD] = "10"
    elif pos_value < pos_threshold and neg_value >=
neg_threshold:
        dict_pred[iD] = "01"
    elif pos_value >= pos_threshold and neg_value >=
neg_threshold:
        dict_pred[iD] = "11"
```

Prima del calcolo finale, i valori presenti nel *gold standard* e quelli ottenuti con ogni singola combinazione di soglie vengono inseriti in due liste (*gold* e *pred*), di modo che il commento a cui fa riferimento la categoria binaria *gold[0]* sia lo stesso a cui fa riferimento *pred[0]*, ugualmente con *gold[1]* e *pred[1]*, *gold[2]* e *pred[2]*, ecc.

```
for key in dict_pred:
        pred.append(dict_pred[key])
        gold.append(dict_gold[key])
```

A questo punto, per valutare l'efficienza di un dato modello di calcolo con una specifica combinazione di soglie, viene calcolata l'*F-measure* (o *F-score*), salvando globalmente quella migliore (insieme alle due relative soglie), tra tutte quelle risultanti dalle possibili combinazioni:

```
output = precision_recall_fscore_support(gold, pred,
average = 'macro')
```

Dove per *F-measure* si intende una media armonica di *Precision (P)* e *Recall (R)*:

$$\frac{2PR}{P+R}$$

Per *Precision* si intende il rapporto tra il numero di *output* corretti, attestati nel *gold standard* (*True Positive, TP)*, e la somma di *output* corretti e errati, non attestati nel *gold standard (False Positive, FP)*:

$$\frac{TP}{TP+FP}$$

Per *Recall*, invece, si intende il rapporto tra il numero di output corretti e la somma di output corretti e mancati (*False Negative, FN)*:

$$\frac{TP}{TP+FN}$$

Il risultato migliore in termini di *F-measure*, tra quelli ottenuti, è di 0.44, con il modello che implementa l al caso 1 di p. 22 utilizzando il lessico ItEM.sorted.polarity.txt.[1] Un risultato basso come quello ricavato dipende dalla strutturazione dello *script* di valutazione: se, ad esempio, ad un determinato commento nel *gold standard* è associata la categoria binaria

[1] Utilizzando invece *average='weighted'* il risultato migliore in termini di *F-measure* è notevolmente superiore, pari a 0.66, con il modello che implementa la formula al caso 1 di p. 22 utilizzando il lessico ANEW.cfg2.bootstrapped.repubblica.itwac.ppmi.cos. Questo perché in tal caso la metrica viene calcolata "for each label, and find their average, weighted by support (the number of true instances for each label). This alters 'macro' to account for label imbalance" (Scikit-learn, *Sk-learn.metrics*), tenuto conto che la maggioranza di commenti presenti nel *gold standard* hanno una polarità negativa.

"10", e l'*output* del modello di calcolo, per lo stesso commento, ha associata la categoria binaria "11", allora il risultato viene trattato alla stregua di un errore. Senza che si tenga conto, in altre parole, di come l'*output* è corretto per metà, poiché la componente di positività è stata predetta dal modello nel modo giusto. È per questo che si è deciso di fare ricorso a una differente modalità di valutazione dell'efficienza dei modelli di calcolo.

5.1. EVALITA 2016 SENTIMENT POLARITY CLASSIFICATION TASK

Per questa seconda fase di valutazione è stato utilizzato uno *script* (v. 9.6) particolarmente confacente alle caratteristiche di questo lavoro, quello della "seconda edizione della campagna di valutazione di sistemi di sentiment analysis (SENTIment POLarity Classification Task), proposta nel contesto di "EVALITA 2016: Evaluation of NLP and Speech Tools for Italian""(Barbieri e altri, 2016a). Il *task* che è stato preso in considerazione ai fini di questo lavoro è il secondo, la *Polarity Classification*, in cui "a system must decide whether a given message is of positive, negative, neutral or mixed sentiment" (2016a). Proprio come nel caso dei modelli di calcolo della presente ricerca, in cui "positive and negative polarities are *not* mutually exclusive and each is annotated as a binary category" (2016a). All'interno di SENTIPOLC 2016 sono presenti anche quattro campi per la rilevazione dell'eventuale presenza di una componente di soggettività e di ironia all'interno, in questo caso, di *tweet*, e altri due che indicano la *literal polarity* (una novità rispetto a SENTIPOLC 2014), "to provide insights into the mechanisms behind polarity shifts in the presence of figurative usage" (2016a).

Lo schema di annotazione è il seguente (Barbieri e altri, 2016b):

```
"id", "subj", "opos", "oneg", "iro", "lpos", "lneg", "top"
```

Nel nostro caso i campi relativi alla soggettività, all'ironia, alla *literal polarity* e all'argomento di riferimento (*top*), non di interesse nel contesto di questo lavoro, sono stati convenzionalmente impostati a "0". Di seguito l'annotazione degli 11 commenti presentati a p. 28 nel formato utilizzato in SENTIPOLC:

```
1461396310541277_1005434672887848,"0","0","1","0","0","0","0"
1424763700871205_1424800487534193,"0","0","1","0","0","0","0"
1537150129632561_1537421186272122,"0","0","0","0","0","0","0"
10154179956375958_10154180112015958,"0","0","1","0","0","0","0"
10154720132141151_1831399290477547,"0","0","0","0","0","0","0"
10154206013596151_10154206408146151,"0","0","1","0","0","0","0"
10154429641327530_10154430630467530,"0","0","0","0","0","0","0"
1528392483841659_1529933777020863,"0","0","1","0","0","0","0"
10154488736667459_10154489479592459,"0","0","1","0","0","0","0"
1513086642038910_1513197738694467,"0","0","1","0","0","0","0"
1533164030030322_1535133223166736,"0","0","1","0","0","0","0"
```

In aggiunta, sono stati annotati manualmente 100 ulteriori commenti rispetto a quelli già presenti nel g*old standard*, in modo da ottenere un risultato più attendibile in fase di valutazione.

Il vantaggio dello *script* di *polarity classification* di EVALITA, che difatti ben si adatta alle esigenze di questo lavoro, risiede in una valutazione indipendente di polarità positiva e negativa, "by computing precision, recall, and F-score for both classes (0 and 1)" (Barbieri e altri, 2016a). Di seguito il criterio di calcolo di *Precision* e *Recall* eseguito dallo *script* di SENTIPOLC (2016a):

$$p_{class}^{pos} = \frac{\#correct_{class}^{pos}}{\#assigned_{class}^{pos}} \qquad r_{class}^{pos} = \frac{\#correct_{class}^{pos}}{\#total_{class}^{pos}}$$

$$p_{class}^{neg} = \frac{\#correct_{class}^{neg}}{\#assigned_{class}^{neg}} \quad r_{class}^{neg} = \frac{\#correct_{class}^{neg}}{\#total_{class}^{neg}}$$

L'*F-score* per le due classi di polarità viene calcolata come la media delle *F-score* delle rispettive coppie:

$$F^{pos} = \frac{(F_0^{pos} + F_1^{pos})}{2} \quad F^{pos} = \frac{(F_0^{pos} + F_1^{pos})}{2}$$

Infine, l'*F-score* globale viene calcolata come la media delle *F-score* delle due classi di polarità.

Ricalcolando l'*F-score* con lo *script* iniziale (v. 5), utilizzando però come riferimento il *gold standard* arricchito di ulteriori 100 commenti, non si sono notati significativi miglioramenti (con average='macro'). Con average='weighted', invece (v. 5), il risultato migliore in termini di *F-score* è aumentato a 0.70 (con lo stesso modello di calcolo e lo stesso lessico).

5.1.1. RISULTATI CON SENTIPOLC

Utilizzando lo *script* di SENTIPOLC si sono notati significativi miglioramenti, in termini di *F-score* globale, rispetto a quelli ottenuti inizialmente. Per ricondurre gli output dei modelli di calcolo allo schema di annotazione di p. 33 sono state utilizzate le due soglie con cui si erano ottenuti i risultati migliori ricalcolando l'*F-score* con il *gold standard* raddoppiato (per ogni modello di calcolo combinato con ciascuno dei tre lessici utilizzati).

Di seguito il riassunto dei risultati ottenuti con SENTIPOLC (le formule di calcolo e i lessici sono numerati con l'ordine in cui sono stati presentati, rispettivamente, a p. 22 e a p. 19):

Modello di calcolo	Pos	Neg	Combined F-score
Formula1 + Lessico1	0.5584	0.5931	0.5758
Formula1 + Lessico2	0.5830	0.5902	**0.5866**
Formula1 + Lessico3	0.5303	0.5992	0.5647
Formula2 + Lessico1	0.5856	0.5571	0.5713
Formula2 + Lessico2	0.5748	0.5674	0.5711
Formula2 + Lessico3	**0.6542**	0.4997	0.5769
Formula3 + Lessico1	0.5541	0.5814	0.5677
Formula3 + Lessico3	0.5500	**0.6094**	0.5797

Tabella1. F-score ottenute con SENTIPOLC

Con il modello che implementa la formula più semplice, quella al caso 1 di p. 22, combinata con il lessico ItEM, è stato ottenuto il risultato migliore in termini di *F-score* globale (0.5866). Un risultato che è in linea con quelli ottenuti in Evalita 2016 SENTIment POLarity Classification Task nel task di polarity classification – che oscillano tra lo 0.5683 e, nel caso migliore, lo 0.6638 (Barbieri e altri, 2016a) – pur derivando da un modello di calcolo molto semplice e che, a differenza di quelli di Evalita, si basa *solo* sul valore emotivo dei termini lessicali, calcolato con il lessico ItEM e senza l'uso di algoritmi di *Machine Learning*.

In conclusione, utilizzando il modello con *F-score* migliore e le relative soglie si è determinata la quantità di commenti (che compongono il corpus) dalla polarità positiva, negativa, neutra e mista. Il risultato è eloquente: la maggioranza (55.3%) dei commenti registrati sulle pagine Facebook degli otto quotidiani presi in esame, infatti, hanno una polarità negativa, mentre il 39.4% non sono connotati emotivamente e i rimanenti hanno associata una polarità positiva o mista (tenuto conto, ovviamente, del margine di errore che il modello di calcolo deve scontare). Ad indicare, forse, come il dibattito politico sul referendum costituzionale abbia *polarizzato* l'opinione pubblica in due tifoserie nettamente definite, con toni spesso duri e talvolta insultanti:

da qui il *sentiment* nella maggioranza dei casi negativo associato ai commenti della popolazione dei lettori di quotidiani su Facebook.

È bene precisare però come un commento dalla polarità negativa non significhi necessariamente, in questo caso, una considerazione a favore del 'No' al referendum; come, d'altra parte, un commento dalla polarità positiva non è obbligatoriamente equivalente a una presa di posizione per il 'Sì'.

A margine è altrettanto opportuno puntualizzare che, partendo da questo risultato, non si possono tracciare fondati paralleli con il risultato percentuale della consultazione referendaria: il campione preso in considerazione – i frequentatori di Facebook e, più nello specifico, quelli che hanno commentato nelle pagine dei quotidiani nell'intervallo di tempo preso in considerazione – non può infatti essere rappresentativo dell'intera popolazione dei votanti al referendum.

Infine, considerando invece globalmente tutti i post che popolano il corpus, la quasi totalità di essi (93%) è costituita da una maggioranza di commenti dalla polarità negativa, mentre il 6,6% sono formati da una maggioranza di commenti dalla connotazione emotiva neutra e lo 0,3% da una maggioranza di commenti dalla polarità mista. Nel corpus non sono presenti, invece, post che abbiano una maggioranza di commenti dalla polarità emotiva positiva.

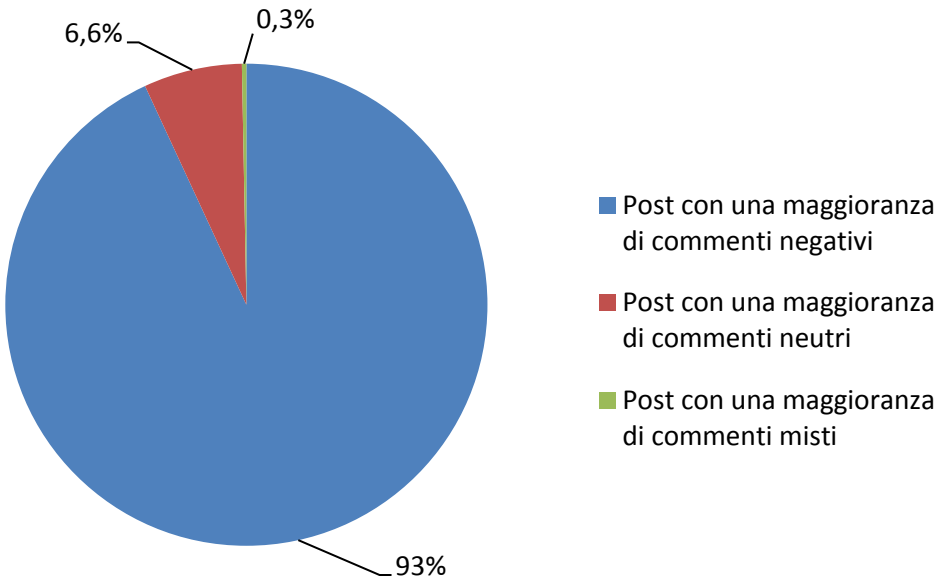

Figura 3. Composizione dei post Facebook che costituiscono il corpus

Esaminando invece la distribuzione dei commenti per singolo quotidiano, la Repubblica è quello che risulta avere la componente maggiore di commenti dalla connotazione emotiva neutra (47,5%), prima del Messaggero (43.4%), mentre nei restanti quotidiani questa componente non supera mai il 40%.

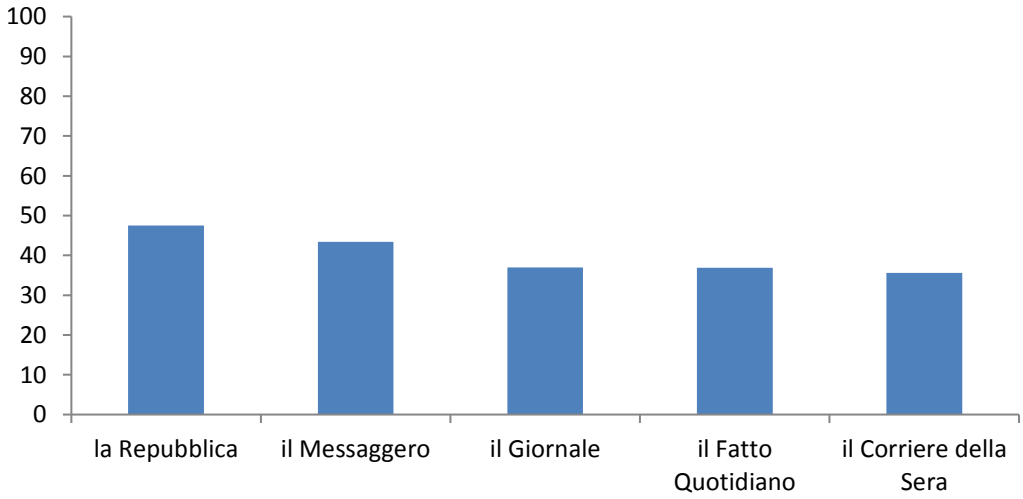

Figura 4. Commenti neutri per quotidiano in percentuale

Diversamente, il quotidiano che annovera la componente più ampia di commenti dalla connotazione emotiva negativa è la Stampa (61.6%), seguita da il Giornale (59.8%), l'Huffingtonpost.it (59,6%), il Corriere della Sera (58%) e il Fatto Quotidiano (57.6%), mentre la Repubblica, al contrario, conta la minore percentuale di commenti emotivamente negativi (48.4%).

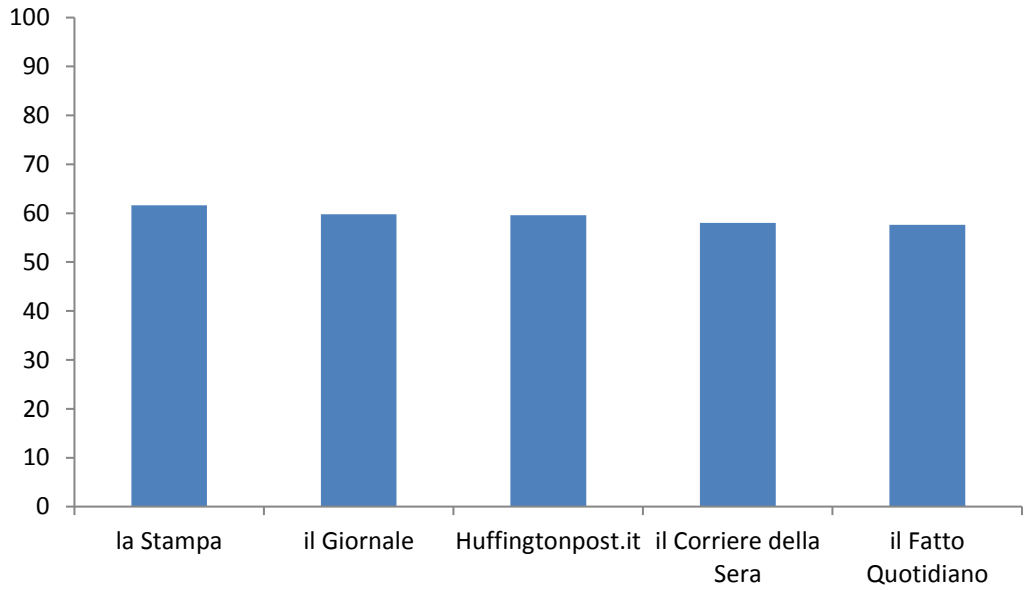

Figura 5. Commenti negativi per quotidiano in percentuale

6. CONCLUSIONI

Gli obiettivi prefissati all'inizio del presente lavoro (v. 1.2) sono stati soddisfatti: da una parte determinare la polarità dei commenti componenti il corpus, dall'altra creare validi modelli di calcolo a tale scopo.

L'efficienza dei modelli di calcolo, infatti, è risultata in linea con lo stato dell'arte in materia di *sentiment analysis*.

Inoltre, la strutturazione dello script di *sentiment classification* di EVALITA 2016, con la possibilità di valutare indipendentemente polarità positiva e negativa, calcolando *Precision, Recall* e *F-score* separatamente per entrambe le classi, messa a confronto con gli script creati appositamente nella ricerca, si è rivelata particolarmente adatta al *task* di questo lavoro.

7. BIBLIOGRAFIA

Barbieri, Francesco, Valerio Basile, Danilo Croce, Malvina Nissim, Nicole Novielli e Viviana Patti. 2016a. *Overview of the Evalita 2016 SENTIment POLarity Classification Task.*

Barbieri, Francesco, Valerio Basile, Danilo Croce, Malvina Nissim, Nicole Novielli e Viviana Patti. 2016b. *Evalita 2016 Sentipolc Task Guidelines.*

Baroni, M., S. Bernardini, F. Comastri, L. Piccioni, A. Volpi, G. Aston e M. Mazzoleni. 2004. *Introducting the La Repubblica Corpus: a large, annotated, TEI(XML)-compliant corpus of newspaper Italian.* In: *Proceedings of the 4th International Conference on Language Resources and Evaluation*, pp. 1171-1174.

Baroni, M., S. Bernardini, A. Ferraresi e E. Zanchetta. 2009. *The wacky wide web: a collection of very large linguistically processed web-crawled corpora.*

Bird, Steven, e Ewan Klein. 2009. *Natural Language Processing with Python.*

Hamdan, Hussam, Patrice Bellot e Frederic Béchet. 2014. *The Impact of Z_score on Twitter Sentiment Analysis*. In: *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014),* pp. 636–641.

Madhavan, Samir. 2015. *Mastering Python for data science*. Packt publishing.

Maynard, Diana, e Kalina Bontcheva. 2016. *Challenges of Evaluating Sentiment Analysis Tools on Social Media*. University of Sheffield.

Pang, Bo, e Lilian Lee. 2002. *Thumbs up? Sentiment classification using Machine Learning Techniques*. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Philadelphia, July 2002*, pp. 79-86.

Pang, Bo, e Lilian Lee. 2008. *Opinion mining and sentiment analysis.* In: *Foundations and trends in Information Retrieval.* vol. 2.

Passaro, Lucia C, e Alessandro Lenci. 2016. *Evaluating context selection strategies to build emotive vector space models.*

Plutchik, Robert. 1994. *The psychology and biology of emotion.* Harper Collins.

Turney, Peter D. 2002. *Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews*. In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics (ACL), Philadelphia, July 2002*, pp. 417-424.

.

8. SITOGRAFIA

Bitext, *Differences between Polarity and Topic-based Sentiment Analysis.* https://blog.bitext.com/polarity-topic-sentiment-analysis.

Docs.Python, *Built-in Functions.* https://docs.python.org/2/library/functions.html.

Facebook for developers. *Token d'accesso.* http://bit.ly/2oe9qrU.

Nlp.Stanford. *Stanford Log-linear Part-Of-Speech Tagger.* https://nlp.stanford.edu/software/tagger.shtml.

Scikit-learn, *Sk-learn.metrics.* http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html.

Unixtimestamp, *Epoch Unix Time Stamp Converter.* http://www.unixtimestamp.com/index.php.

Wikipedia, *Scarto quadratico medio.* https://it.wikipedia.org/wiki/Scarto_quadratico_medio.

9. APPENDICE

9.1. SCRIPT DI FILTRAGGIO DEL CORPUS

```
import os
import string
#definizioneFunzioni

#definizione funzione append post e commenti figli al file di
output globale
def FileSave(filename,content):
   with open(filename, "a") as myfile:
   myfile.write(content)
#definizione funzione lettura ribuffer e append al file di
output globale
def lavora_ribuffer(namebuffer,filenameout):
   with open(namebuffer, "r") as content:
   verocontent = content.read()
   FileSave(filenameout,verocontent)
#corpoProgramma
os.chdir("Scrivania/Cocchi")
cwd = os.getcwd()
for file in os.listdir(cwd):
   if file.endswith(".pos"):
       in_file = open(file, "r")
       preleva_post = False
       ribuffer = open("filebuffer.txt", "wb")
       words = []
       for line in in_file:
          if line.startswith("<doc") and 'type="post"' in
line:
              type = "post"
              if preleva_post == True:
                 print "trovato"
                 ribuffer.close()
                 lavora_ribuffer("filebuffer.txt",
"Output.txt")
                 preleva_post = False
                 ribuffer = open("filebuffer.txt", "wb")
                 #azzera words
                 words = []
              else:
                 ribuffer.close()
                 ribuffer=open("filebuffer.txt","wb")
          else:
             if line.startswith("</doc>") and type == "post":
```

```
            type = "comment"
            #controlla words e setta preleva_post
            for i,j in zip(words, words[1:]):
               if ((i == "referendum" or i == "Referendum")
and j == "costituzionale") or ((i == "riforma" or i ==
"Riforma") and (j == "costituzionale" or j == "Boschi")) or
((i=="ddl" or i == "Ddl") and (j=="Boschi" or j ==
"costituzionale")) or (i == "4" and j == "dicembre") or (i ==
"quesito" and j == "referendario"):
                  preleva_post=True
            elif type == "post":
               sequenza_parole=line.split( )
               if len(sequenza_parole) != 0:
               parola_colonna_3=sequenza_parole[2]
               words.append(parola_colonna_3)

         ribuffer.write(line)
      else:
         if preleva_post == True:
            lavora_ribuffer("filebuffer.txt","Output.txt")
      in_file.close()
```

9.2. SCRIPT DEL PRIMO MODELLO DI CALCOLO

```
from collections import defaultdict
lexicon_dict = defaultdict(lambda:[0,0])
def CaricaLessico(lessico):
   with open(lessico) as lexicon:
      for line in lexicon:
         sequenza = line.split( )
         parola = sequenza[1]
         valore = sequenza[2].replace(',','.')
         if "positive" in line:
            lexicon_dict[parola][0] = valore
         elif "negative" in line:
            lexicon_dict[parola][1] = valore
CaricaLessico("Scrivania/Lessici/ANEW.cfg2.bootstrapped.FB-
NEWS15.ppmi.cos")
in_file = open("Scrivania/Cocchi/Output.txt", "r")
tipo = "post"
contatore = 0
valore_emotivo = 0
pol_pos = 0
pol_neg = 0
ID = 'id="1533164030030322_1533359486677443"'
for line in in_file:
```

```
   if line.startswith("<doc") and 'type="comment"' in line:
      sequence = line.split( )
      ID_now = sequence[2]
      tipo = "comment"
      print "COMMENT"
      if contatore > 0:
         with
open("Scrivania/Lessici/Emotional_Sum_wordbyword_FB.txt", "a")
as output:
            output.write(ID + ": " + "pol_pos:" +
repr(pol_pos) + " " + "pol_neg:" + repr(pol_neg) + " " +
"aggr:" + repr(valore_emotivo) + "\n")
            ID = ID_now
            print contatore
            valore_emotivo = 0
            pol_pos = 0
            pol_neg = 0
            print valore_emotivo
   elif line.startswith("</doc>") and tipo == "comment":
      contatore = contatore + 1
   elif line.startswith("<doc") and 'type="post"' in line:
      tipo = "post"
      print "POST"
   elif tipo == "comment":
      if line[0].isdigit():
         sequenza = line.split( )
         parola = sequenza[2]
         PoS = sequenza[3].lower()
         true_parola = parola + "-" + PoS
         positive = float(lexicon_dict[true_parola][0])
         negative = float(lexicon_dict[true_parola][1])
         valore_emotivo = valore_emotivo + positive - negative
         pol_pos = pol_pos + positive
         pol_neg = pol_neg + negative

with open("Scrivania/Lessici/Emotional_Sum_wordbyword_FB.txt",
"a") as output:
   output.write(ID + ": " + "pol_pos:" + repr(pol_pos) + " " +
"pol_neg:" + repr(pol_neg) + " " + "aggr:" +
repr(valore_emotivo))
   print valore_emotivo
   print contatore

in_file.close()
```

9.3. SCRIPT DEL SECONDO MODELLO DI CALCOLO

```
from collections import defaultdict
lexicon_dict = defaultdict(lambda:[0,0])
def CaricaLessico(lessico):
   with open(lessico) as lexicon:
      for line in lexicon:
         sequenza = line.split( )
         parola = sequenza[1]
         valore = sequenza[2].replace(',','.')
         if "positive" in line:
            lexicon_dict[parola][0] = valore
         elif "negative" in line:
            lexicon_dict[parola][1] = valore
CaricaLessico("Scrivania/Lessici/ANEW.cfg2.bootstrapped.FB-
NEWS15.ppmi.cos")
in_file = open("Scrivania/Cocchi/Output.txt", "r")
tipo = "post"
contatore = 0
valore_positivo = 0
contatore_pos = 0
valore_negativo = 0
contatore_neg = 0
ID = 'id="1533164030030322_1533359486677443"'
for line in in_file:
   if line.startswith("<doc") and 'type="comment"' in line:
      sequence = line.split( )
      ID_now = sequence[2]
      tipo = "comment"
      print "COMMENT"
      if contatore > 0:
         with
open("Scrivania/Lessici/Emotional_Sum_average_FB.txt", "a") as
output:
            if contatore_pos > 0:
               media_pos = valore_positivo/contatore_pos
               media_neg = valore_negativo/contatore_neg
               risultato = media_pos - media_neg
               output.write(ID + ": " + "pol_pos:" +
repr(media_pos) + " " + "pol_neg:" + repr(media_neg) + " " +
"aggr:" + repr(risultato) + "\n")
            else:
               output.write(ID + ": pol_pos:0 pol_neg:0
aggr:0" + "\n")
            ID = ID_now
            print contatore
```

```
            valore_positivo = 0
            valore_negativo = 0
            contatore_pos = 0
            contatore_neg = 0
   elif line.startswith("</doc>") and tipo == "comment":
      contatore = contatore + 1
   elif line.startswith("<doc") and 'type="post"' in line:
      tipo = "post"
      print "POST"
   elif tipo == "comment":
      if line[0].isdigit():
         sequenza = line.split( )
         parola = sequenza[2]
         PoS = sequenza[3].lower()
         true_parola = parola + "-" + PoS
         positive = float(lexicon_dict[true_parola][0])
         contatore_pos = contatore_pos + 1
         negative = float(lexicon_dict[true_parola][1])
         contatore_neg = contatore_neg + 1
         valore_positivo = valore_positivo + positive
         valore_negativo = valore_negativo + negative

with open("Scrivania/Lessici/Emotional_Sum_average_FB.txt",
"a") as output:
   media_pos = valore_positivo/contatore_pos
   media_neg = valore_negativo/contatore_neg
   risultato = media_pos - media_neg
   output.write(ID + ": " + "pol_pos:" + repr(media_pos) + " "
+ "pol_neg:" + repr(media_neg) + " " + "aggr:" +
repr(risultato))
   print contatore

in_file.close()
```

9.4. SCRIPT DEL TERZO MODELLO DI CALCOLO

```
from collections import defaultdict
lexicon_dict = defaultdict(lambda:[0,0])
def CaricaLessico(lessico):
   with open(lessico) as lexicon:
      for line in lexicon:
         sequenza = line.split( )
         parola = sequenza[1]
         valore = sequenza[2].replace(',','.')
         if "positive" in line:
            lexicon_dict[parola][0] = valore
         elif "negative" in line:
```

```
            lexicon_dict[parola][1] = valore
CaricaLessico("Scrivania/Lessici/ANEW.cfg2.bootstrapped.repubb
lica.itwac.ppmi.cos")
in_file = open("Scrivania/Cocchi/Output.txt", "r")
tipo = "post"
contatore = 0
valore_emotivo = 0
pol_pos = 0
pol_neg = 0
ID = 'id="1533164030030322_1533359486677443"'
for line in in_file:
   if line.startswith("<doc") and 'type="comment"' in line:
      sequence = line.split( )
      ID_now = sequence[2]
      tipo = "comment"
      print "COMMENT"
      if contatore > 0:
         with
open("Scrivania/Lessici/Emotional_Sum_normalized_repubblica.tx
t", "a") as output:
            output.write(ID + ": " + "pol_pos:" +
repr(pol_pos) + " " + "pol_neg:" + repr(pol_neg) + " " +
"aggr:" + repr(valore_emotivo) + "\n")
            ID = ID_now
            print contatore
            valore_emotivo = 0
            pol_pos = 0
            pol_neg = 0
            print valore_emotivo
   elif line.startswith("</doc>") and tipo == "comment":
      contatore = contatore + 1
   elif line.startswith("<doc") and 'type="post"' in line:
      tipo = "post"
      print "POST"
   elif tipo == "comment":
      if line[0].isdigit():
         sequenza = line.split( )
         parola = sequenza[2]
         PoS = sequenza[3].lower()
         true_parola = parola + "-" + PoS
         positive = float(lexicon_dict[true_parola][0])
         negative = float(lexicon_dict[true_parola][1])
         somma = positive + negative
         if somma > 0:
            true_positive = positive/somma
            true_negative = negative/somma
            valore_emotivo = valore_emotivo + true_positive -
true_negative
```

```
            pol_pos = pol_pos + true_positive
            pol_neg = pol_neg + true_negative

with
open("Scrivania/Lessici/Emotional_Sum_normalized_repubblica.tx
t", "a") as output:
   output.write(ID + ": " + "pol_pos:" + repr(pol_pos) + " " +
"pol_neg:" + repr(pol_neg) + " " + "aggr:" +
repr(valore_emotivo))
   print valore_emotivo
   print contatore

in_file.close()
```

9.5. SCRIPT DI VALUTAZIONE DELL'EFFICIENZA

```
from collections import defaultdict
from sklearn.metrics import precision_recall_fscore_support
import numpy as np
import itertools

dict_gold = defaultdict(lambda:[0])
with open("Scrivania/Gold.txt", "r") as gold:
    for line in gold:
        sequenza = line.split()
        id_comment = sequenza[0]
        true_id = id_comment[:-1]
        dict_gold[true_id] = sequenza[1]

dict_pred = defaultdict(lambda:[0])
gold = []
pred = []

def SalvaPol(pos_threshold, neg_threshold, pos_value,
neg_value, iD):
    if pos_value < pos_threshold and neg_value <
neg_threshold:
        dict_pred[iD] = "00"
    elif pos_value >= pos_threshold and neg_value <
neg_threshold:
        dict_pred[iD] = "10"
    elif pos_value < pos_threshold and neg_value >=
neg_threshold:
        dict_pred[iD] = "01"
    elif pos_value >= pos_threshold and neg_value >=
neg_threshold:
        dict_pred[iD] = "11"
```

```
massimo_pos = 0
massimo_neg = 0
minimo_pos = 1
minimo_neg = 1
dict_document = defaultdict(lambda:[0,0])
with
open("Scrivania/Lessici/Emotional_Sum_normalized_repubblica.tx
t", "r") as documento:
    for line in documento:
        sequence = line.split()
        pos = float(sequence[1][8:])
        neg = float(sequence[2][8:])
        vero_id = sequence[0][:-1]
        if vero_id in dict_gold:
            dict_document[vero_id] = [pos, neg]
        if pos > massimo_pos:
            massimo_pos = pos
        elif pos < minimo_pos:
            minimo_pos = pos
        elif neg > massimo_neg:
            massimo_neg = neg
        elif neg < minimo_neg:
            minimo_neg = neg
best_pos = 0
best_neg = 0
f_measure = 0
contatore = 0
soglie_pos = np.linspace(minimo_pos, massimo_pos, 401)
soglie_neg = np.linspace(minimo_neg, massimo_neg, 401)
combinations = list(itertools.product(soglie_pos, soglie_neg))
for element in combinations:
    for key in dict_document:
        pol_pos = float(dict_document[key][0])
        pol_neg = float(dict_document[key][1])
        SalvaPol(element[0], element[1], pol_pos, pol_neg,
key)
    for key in dict_pred:
        pred.append(dict_pred[key])
        gold.append(dict_gold[key])
    output = precision_recall_fscore_support(gold, pred,
average = 'macro')
    contatore = contatore + 1
    f_score = output[2]
    if f_score > f_measure:
        f_measure = f_score
        best_pos = element[0]
        best_neg = element[1]
    print contatore
    pred = []
    gold = []
print f_measure, best_pos, best_neg
```

9.6. SCRIPT DI SENTIPOLC

```
#!/usr/bin/env python
# evaluation script for the SENTIPOLC 2016 shared task

verbose=True

from argparse import ArgumentParser

argparser = ArgumentParser(description='')
argparser.add_argument('-r', dest='result_file',
action='store', help='CSV file of the run results')
argparser.add_argument('-g', dest='gold_file', action='store',
default="sentipolc16_gold2000.csv", help='gold standard
annotation CSV file')
args = argparser.parse_args()

# read gold standard and populate the count matrix
gold = dict()
gold_counts =  {'subj':{'0':0,'1':0},
                'opos':{'0':0,'1':0},
                'oneg':{'0':0,'1':0},
                'iro':{'0':0,'1':0},
                'lpos':{'0':0,'1':0},
                'lneg':{'0':0,'1':0}
                }
with open(args.gold_file) as f:
    for line in f:
        if len(line.split(',')) > 6:
            id, subj, opos, oneg, iro, lpos, lneg, top =
map(lambda x: x[1:-1], line.rstrip().split(','))
        else:
            id, subj, opos, oneg, iro, top = map(lambda x:
x[1:-1], line.rstrip().split(','))
        #id, subj, opos, oneg, iro, lpos, lneg, top =
map(lambda x: x[1:-1], line.rstrip().split(','))
        gold[id] = {'subj':subj, 'opos':opos, 'oneg':oneg,
'iro':iro, 'lpos':lpos, 'lneg':lneg}
        gold_counts['subj'][subj]+=1
        gold_counts['opos'][opos]+=1
        gold_counts['oneg'][oneg]+=1
        gold_counts['iro'][iro]+=1

        if len(line.split(',')) > 6:
```

```
            gold_counts['lpos'][lpos]+=1
            gold_counts['lneg'][lneg]+=1


# read result data
result = dict()
with open(args.result_file) as f:
    for line in f:
        if len(line.split(',')) > 6:
            id, subj, opos, oneg, iro, lpos, lneg, top =
map(lambda x: x[1:-1], line.rstrip().split(','))
            result[id]= {'subj':subj, 'opos':opos,
'oneg':oneg, 'iro':iro}
        else:
            id, subj, opos, oneg, iro, top = map(lambda x:
x[1:-1], line.rstrip().split(','))
            result[id]= {'subj':subj, 'opos':opos,
'oneg':oneg, 'iro':iro}

# evaluation: single classes
for task in ['subj', 'opos', 'oneg', 'iro']:    #add 'lpos'
and 'lneg' if you want to measure literal polairty
    # table header
    if verbose: print "\ntask: {}".format(task)
    if verbose: print "prec. 0\trec. 0\tF-sc. 0\tprec. 1\trec.
1\tF-sc. 1\tF-sc."
    correct =  {'0':0,'1':0}
    assigned = {'0':0,'1':0}
    precision ={'0':0.0,'1':0.0}
    recall =   {'0':0.0,'1':0.0}
    fscore =   {'0':0.0,'1':0.0}

    # count the labels
    for id, gold_labels in gold.iteritems():
        if (not id in result) or result[id][task]=='':
            pass
        else:
            assigned[result[id][task]] += 1
            if gold_labels[task]==result[id][task]:
                correct[result[id][task]] += 1

    # compute precision, recall and F-score
    for label in ['0','1']:
        try:
            precision[label] =
float(correct[label])/float(assigned[label])
            recall[label] =
float(correct[label])/float(gold_counts[task][label])
```

```
            fscore[label] = (2.0 * precision[label] *
recall[label]) / (precision[label] + recall[label])
        except:
            # if a team doesn't participate in a task it gets
default 0 F-score
            fscore[label] = 0.0

    # write down the table
    print
"{0:.4f}\t{1:.4f}\t{2:.4f}\t{3:.4f}\t{4:.4f}\t{5:.4f}\t{6:.4f}
".format(
            precision['0'], recall['0'], fscore['0'],
            precision['1'], recall['1'], fscore['1'],
            (fscore['0'] + fscore['1'])/2.0)


# polarity evaluation needs a further step
if verbose: print "\ntask: polarity"
if verbose: print "Combined F-score"
correct =  {'opos':{'0':0,'1':0}, 'oneg':{'0':0,'1':0}}
assigned = {'opos':{'0':0,'1':0}, 'oneg':{'0':0,'1':0}}
precision ={'opos':{'0':0.0,'1':0.0},
'oneg':{'0':0.0,'1':0.0}}
recall =   {'opos':{'0':0.0,'1':0.0},
'oneg':{'0':0.0,'1':0.0}}
fscore =   {'opos':{'0':0.0,'1':0.0},
'oneg':{'0':0.0,'1':0.0}}

# count the labels
for id, gold_labels in gold.iteritems():
    for cl in ['opos','oneg']:
        if (not id in result) or result[id][cl]=='':
            pass
        else:
            assigned[cl][result[id][cl]] += 1
            if gold_labels[cl]==result[id][cl]:
                correct[cl][result[id][cl]] += 1

# compute precision, recall and F-score
for cl in ['opos','oneg']:
    for label in ['0','1']:
        try:
            precision[cl][label] =
float(correct[cl][label])/float(assigned[cl][label])
            recall[cl][label] =
float(correct[cl][label])/float(gold_counts[cl][label])
```

```
            fscore[cl][label] = float(2.0 *
precision[cl][label] * recall[cl][label]) /
float(precision[cl][label] + recall[cl][label])
        except:
            fscore[cl][label] = 0.0

fscore_pos = (fscore['opos']['0'] + fscore['opos']['1'] ) /
2.0
fscore_neg = (fscore['oneg']['0'] + fscore['oneg']['1'] ) /
2.0

# write down the table
print "{0:.4f}".format((fscore_pos + fscore_neg)/2.0)
```