Distanze di amicizia e profili di similarità tra utenti twitter

Riccardo La Grassa 0647323

Abstract

L'obiettivo della prima parte del progetto è stato quello di fornire una mappatura parziale di un sottoinsieme dell'Universo Twitter, con lo scopo di trovare distanze di amicizia entro i limiti del valore ipotizzato dalla teoria dei sei gradi di separazione. La seconda parte prevede l'utilizzo ed il confronto di tre modelli utilizzati per il calcolo della similarità tra utenti twitter, in particolare si introduce il modello vettoriale supportato dal database wordnet. Vengono presentati anche due 'Approcci alla classificazione', uno basato su una struttura ad albero di istanze appartenenti ad un dato gruppo di concetti, e l'altro sul classico modello vettoriale supportato da categorie etichettate da un essere umano (supervisionato). Sono state analizzate nel dettaglio diverse coppie di utenti, etichettati 22 utenti twitter con risultati apprezzabili e condotti numerosi test al fine di verificare la correttezza dei risultati.

I. Introduzione

La fase preliminare del progetto richiama a gran voce una teoria degli anni '20. La teoria dei sei gradi di separazione, un'ipotesi secondo la quale qualunque entità, attraverso una rete di conoscenze e relazioni può essere interconnessa ad un altra con non più di sei intermediari.
Formalmente, dato un insieme di N persone, qual è la probabilita che ogni membro di N sia connesso ad un altro membro attraverso k_1,k_2,k_3,...k_n collegamenti?
Un esperimento condotto da un gruppo di informatici italiani nel 2011 dimostrò che nel mondo di facebook, il 92% delle coppie di target è separato da non più di 4 gradi.
Sebbene esistono esperimenti condotti nel 2006 da parte della Microsoft che all'interno dello storico MSN Messenger i gradi di separazione erano in media 6.6 (comuque sempre all'interno dei sei gradi ipotizzati).
Il modello generale su cui è basato l'intero progetto, viene schematizzato in tre fasi: Data Preparation, Analysis, Visualization.
Si è scelto di creare un formato che potesse essere compreso per la generazione nelle fasi successive, di un grafo utilizzando le librerie di (pygraphml). Ho scelto di adattare il formato con la suddetta libreria per poter utilizzare un software grafico di nome cytoscape per la rappresentazione visiva (e notevole) dei grafi generati.

Formato creato:
nodo:[following0,following1,.....,following n]

La prima parte progettuale è stato suddivisa in tre moduli che saranno descritti in dettaglio a seguire.

Creazioni livelli di profondità ed acquisizione dei dati

Il codice in basso, mostra chiaramente nelle fasi iniziali, l'acquisizione dei following del nodo considerato al primo livello di profondità. Nelle fasi successive, sulla base dei nodi già considerati in precedenza, si recuperano le

informazioni sui nodi figli di ognuno. Procedendo iterativamente si ottengono I following, dei following dei following dei ….

```
first_level.append(get_following(get_name_account))  #first level
for i in range(0, len(first_level)):
    for j in range(0, len(first_level[i])):
        second_level.append(get_following(first_level[i][j]))

for i in range(0, len(second_level)):
    for j in range(0, len(second_level[i])):
        thirt_level.append(get_following(second_level[i][j]))

for i in range(0, len(thirt_level)):
    for j in range(0, len(thirt_level[i])):
        fourth_level.append(get_following(thirt_level[i][j]))
```

```
def get_following(node):
    while True:
        try:
            users = tweepy.Cursor(api.friends_ids, id=node, count=n_node).pages(1)
            l = list()
            for i in users:
                for user in i:
                    l.append(user)
            return l
        except tweepy.TweepError as e:
            if ('88' in e.reason):
                print('Wait: 15 min\n')
                time.sleep(60 * 15)
            else:
                print('I cannot to get data from one account\nTweet are protected\n')
                return l
```

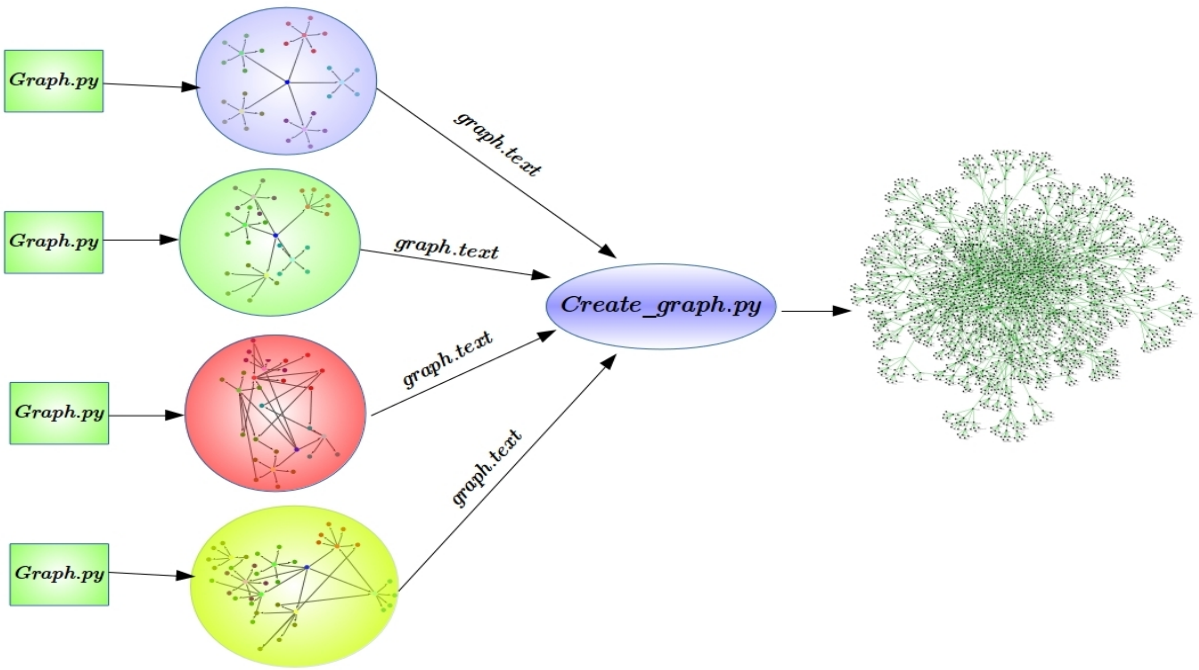
Creazione del formato

```
    with open(name, 'w') as f:
        for item in first_l:
            f.write(get_name_account + ':' + "%s\n" % item)

        i = 0
        for item in second_l:
            f.write(str(first_l[0][i]) + ':' + "%s\n" % item)
            i += 1
        i = 0

        j_index = 0
        for j in range(0, len(second_l)):
            for i in range(0, len(second_l[j])):
                f.write(str(second_l[j][i]) + ':' + "%s\n" % thirt_l[j_index])
                j_index += 1

        j_index = 0
        for j in range(0, len(thirt_l)):
            for i in range(0, len(thirt_l[j])):
                f.write(str(thirt_l[j][i]) + ':' + "%s\n" % forth_l[j_index])
                j_index += 1
```

Tempi di ritardo e scorciatoie

Il seguente link https://dev.twitter.com/rest/public/rate-limits mostra la tabella delle chiamate max che si possono effettuare nell'arco di 15 minuti. Questa vincolo è letale per una mappatura globale dell'universo twitter. Nella parte progettuale sono state largamente utilizzate le api messe a disposizione per gli sviluppatori per acquisire dati da Twitter. nella fase preliminare del progetto, adempiendo al primo punto richiesto, si è utilizzata sopratutto la funzione per acquisire i following di uno specifico utente. Twitter inpone max 15 chiamate in 15 minuti per questa specifica. Di conseguenza, partendo da un utente X, e considerando soltanto 4 livelli di profondità ed 7 following per users, il software impiegherà $7^3+7^2+7^1+1$ minuti = 6,66 ore.

Le ragioni di questo blocco saranno per evitare questo genere di mappatura, o per evitare possibili bot e quindi situazioni di saturazione della rete. Per poter utilizzare le api di twitter occore la registrazione per sviluppatore e la creazione di token (chiavi). Le credenziali di utilizzo per le api sono valide solo per l'app di rete creata sul sito dev twitter. il sistema permette la creazione di diverse app con diverse credenziali d'accesso per le api. Pertanto, considerando il blocco imposto da twitter (ragionevole) si è optato di creare diverse app twitter con relative credenziali in modo da ridurre drasticamente i tempi di acquisizione dei dati. La figura in basso rappresenta l'esecuzione dello stesso file 'graph.py' eseguito su macchine diverse e su target diversi. I risultati saranno diversi grafi sulla base del target relativo. Il file create_graph.py relazionato col precedente sono stati pensati e creati per aumentare l'acquisizione dei dati e ridurre i tempi dettati dal blocco twitter. Create_graph.py elimina le ridondanze e crea un grafico univoco con relazioni uniche tra tutti i nodi. Infine esporta il grafo in formato graphml (xml)
In basso lo Scenario:



```
c=first_level[0][0].split(':')
g.set_root_by_attribute(c[0])

for i in range(0,len(first_level)):
    c=first_level[i][0].split(':')
    split_children = c[1].replace('[', '').replace(']', '').replace(',', '')
    resplit = split_children.split()

    flag=False
    for node in g.nodes():
        if c[0] == node['label']:
            flag = True
            break
    if not flag:
        g.add_node(c[0]) #if node not exist add it

    for i in range(0, len(resplit)):
        flag=False
        for node in g.nodes():
            if resplit[i] == node['label']:
                flag=True
                break
        if not flag:
            g.add_node(resplit[i]) #add node (in this case i see his children)

        flag = False
        for edge in g.edges():
            if resplit[i] == edge.node2['label'] and c[0] == edge.node1['label']:
                flag = True
                break
        if not flag:
            g.add_edge_by_label(c[0], resplit[i])  # add edge (if not exist!)
```

Il codice in alto contenuto nel file create_graph.py, come già accennato prima , acquisisce tutte le informazioni dei formati creati da più istanze

(graph.py) fondendole in un unico grafo eliminando nodi e archi già presenti (Merge)

Find it!

Il terzo modulo della prima parte del progetto, chiamato find_it.py ha lo scopo di trovare la distanza tra il target ed un nodo di partenza (root).
Tuttavia, per poter trovare risultati significativi, la quantità di dati recuperati dal software (per via del blocco twitter) è limitata, considerando anche l'avvio di più istanze dello stesso modulo. Il grafo generato è formato da 2746 nodi e 3024 archi. Un ottima comparazione, è stata quella di avviare il modulo "find it.py" su quest'ultimo grafo (che chiamerò small graph) ed un altra istanza eseguita su un grafo molto più grande (huge graph). Il dataset messo a disposizione dallo Standford University fa al caso nostro. Per eseguire una corretta comparazione, mediante un modulo supplementare python creato appositamente, si è eseguita un operazione di merge tra I dati recuperati dal software ed I dati dello standford. I risultato finale è un grafo con 13020 nodi e 195886 archi. L'immagine in basso mostra chiaramente la comparazione fra l'esecuzione del modulo su entrambi I grafi, e dimostra che la 'distanza' nonchè la profondità del nodo rispetto al root diminuisce con l'aumentare della rete di conoscenza.

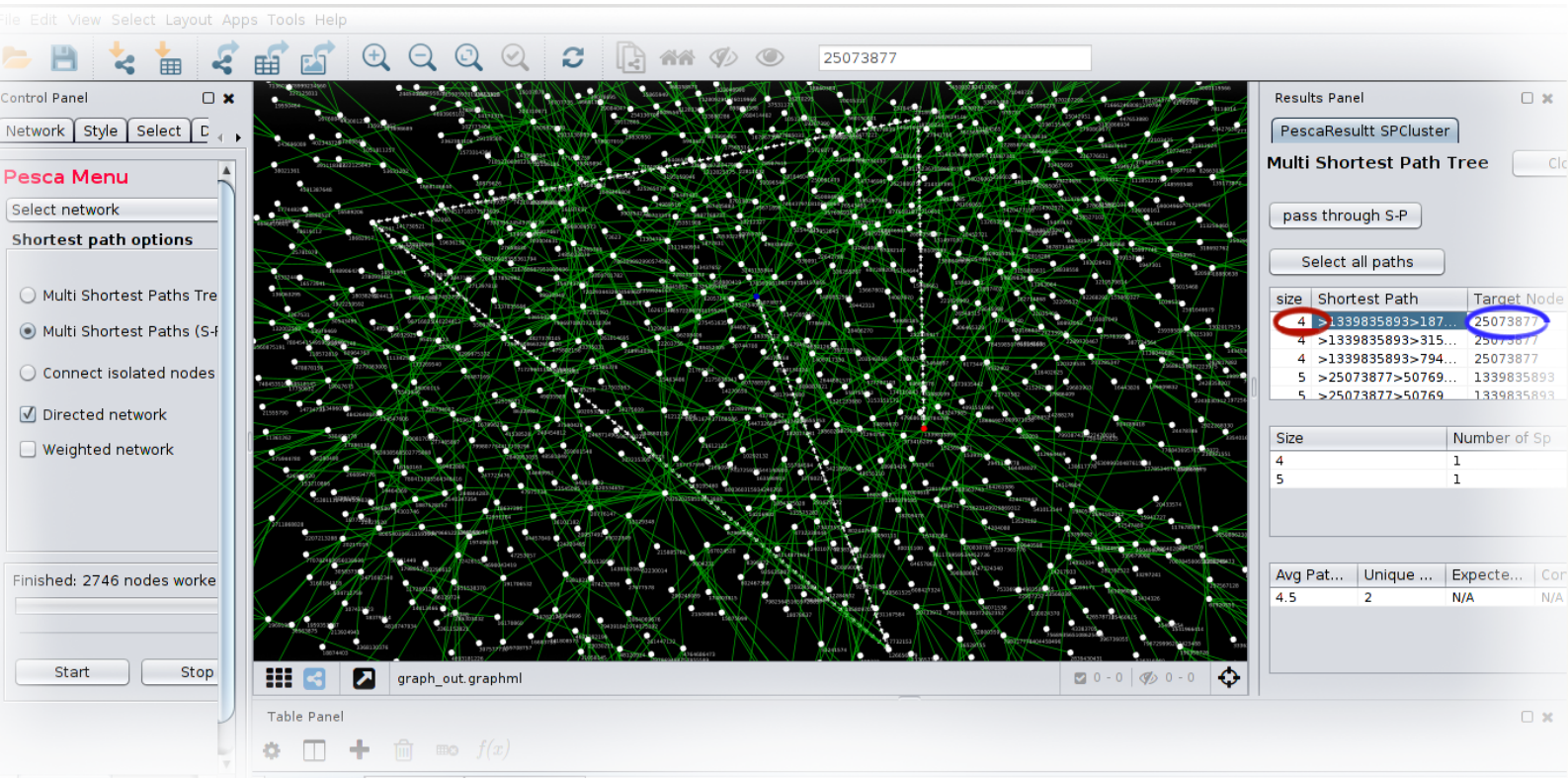
Analysis Data				
	Root	**Target**	**Depth**	**Time**
Small graph	*SnoopDogg*	*_damarisjoy*	10	0.025
Huge graph			**9**	9.062
Small graph	*SnoopDogg*	*TomiLahren*	10	0.026
Huge graph			**9**	9.099
Small graph	*poeticrat*	*VTJuiceCo*	12	0.0162
Huge graph			12	1.84
Small graph	*EastlandCVAC*	*HillaryClinton*	**null**	-
Huge graph			**12**	9.27
Small graph	*SnoopDogg*	*RakeshAgrawal*	10	0.026
Huge graph			**9**	9.288

Legend		
	Nodes	**Edges**
Small graph	2746	3024
Huge graph	**13020**	**195886**

Come si evince dalla parte contrassegnata in rosso(immagine in basso), I risultati del modulo del progetto e del software cytoscape, usato per la rappresentazione grafica dei dati sono perfettamente allineati.

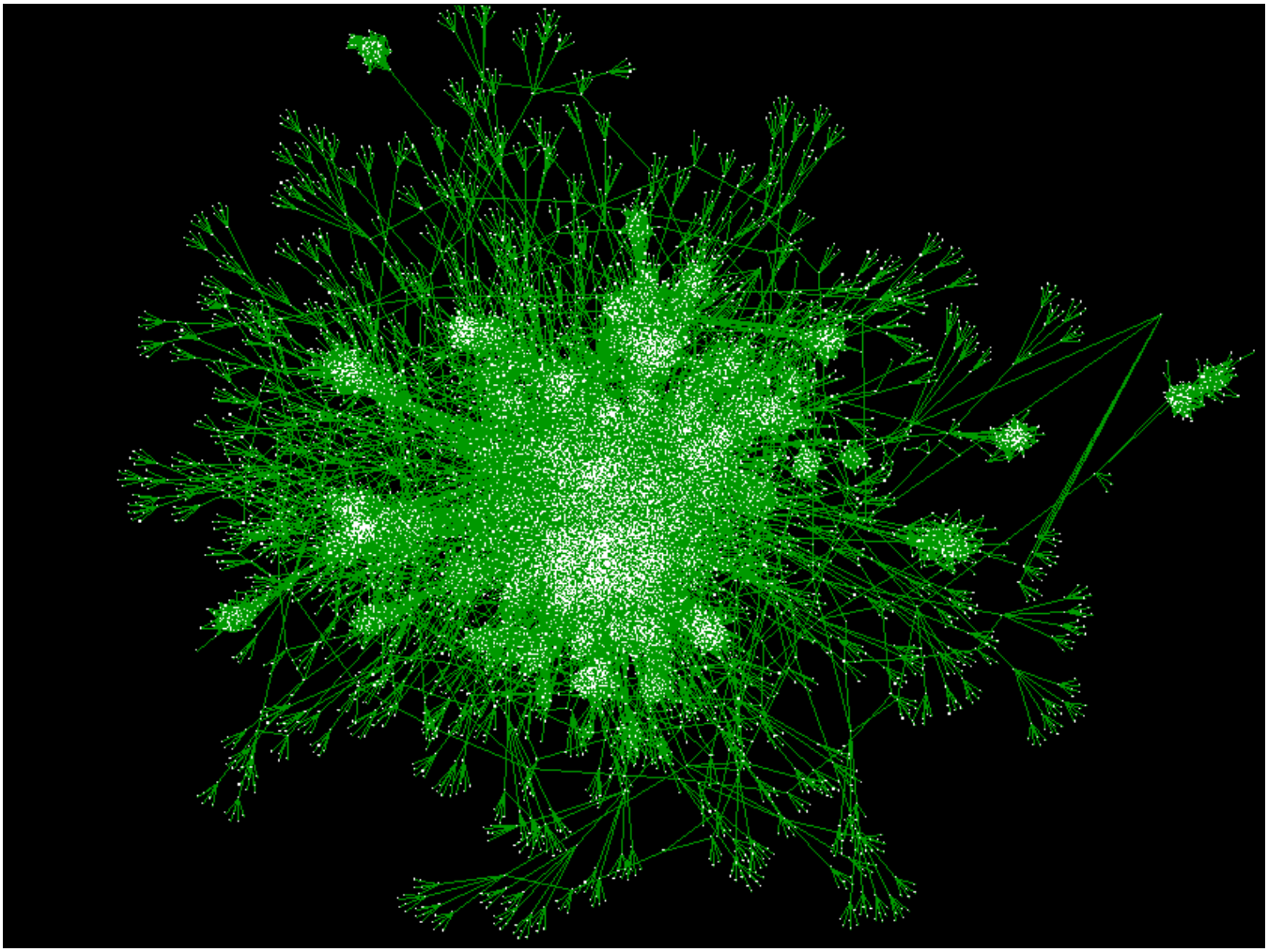
Risultati del modulo find_it.py (root=Hillary Clinton, target=Trump)

Target FOUND!
Name: realDonaldTrump
Depth: 4
Time: 0.0013377666473388672



In basso, viene mostrata la parte centrale del modulo find it.py. Essa scansione il grafo partendo dalla radice e procedendo per profondità fino a trovare il nodo target. Viene considerata un ulteriore lista che contiene I nodi già visitati ed un variabilie 'flag' che funge da frontiera viene introdotta ogni volta che si scende di livello.

```
def bfs_search(root,id_target):
    l=[]
    visited=[]
    l.append(root)
    visited.append(root)
    l.append('flag')
    depth=1
    start=time.time()
    while len(l) > 1:
        x = l.pop(0)
        if x != 'flag':
            for child in g.children(x):
                if (child['label'] == id_target):
                    print('Target FOUND!\nName: ', sys.argv[2])
                    print('Depth: ',depth,'\nTime: ',time.time() - start)
                    return
                if not (child in visited):
                    l.append(child)
                    visited.append(child)
        else:
            depth += 1
            l.append('flag')
print('Not Found')
```

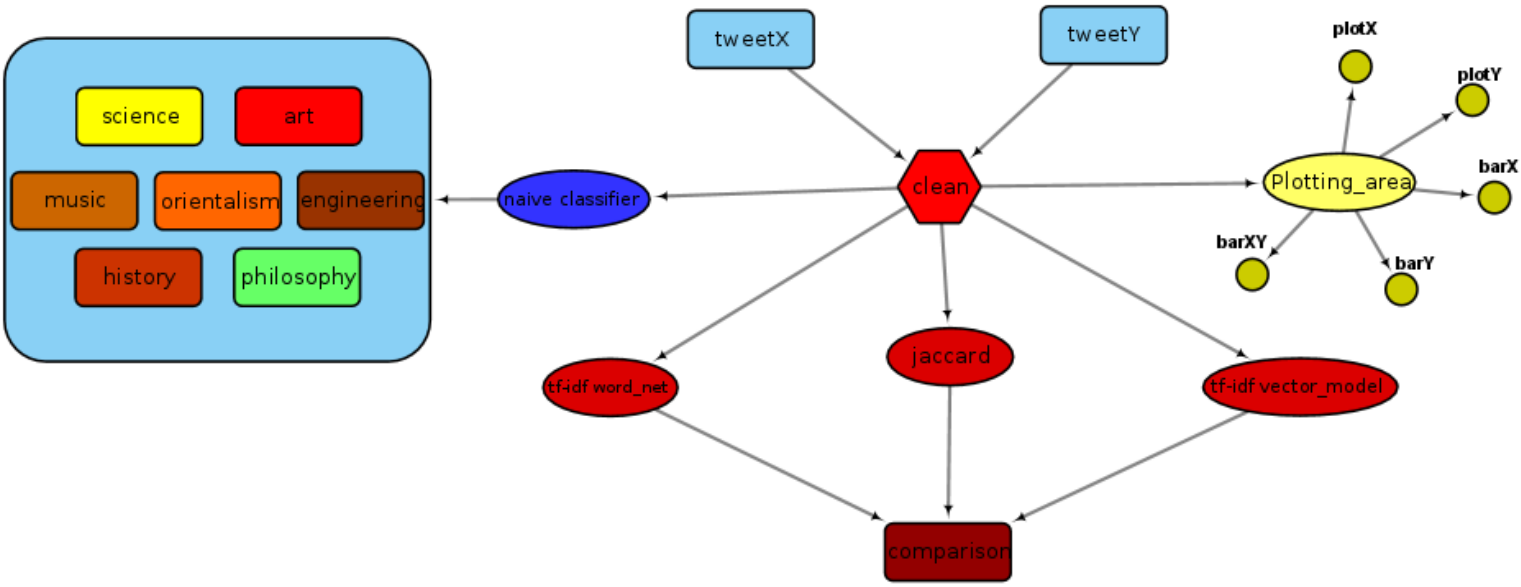


In alto, il grafo ottenuto tra la fusione dei dati ricavati in fase progettuale ed I dati dello Standford University

2 parte

II. Introduzione

La seconda fase del progetto è schematizzata nella figura in basso.



Viene suddivisa in moduli:acquisizione dati e cleaning,classificatore,similarità con coefficienti di jaccard e modello vettoriale,modello con wordnet,area di plotting. Verranno analizzati in dettaglio i vari moduli creati nel progetto ed i loro relativi risultati in fase di testing.

Acquisizione dati

L'operazione ha richiesto un largo utilizzo delle api messe a disposizione da twitter per gli sviluppatori, relazionate a delle credenziali vincolate per il

loro accesso. Come già accennato nella parte 1 del progetto, anche questa fase deve sottostare alle politiche di twitter (ragionevoli per prevenire intasamenti delle linee ai loro server). Il max numero di post ricevuti per un utente X è di 3220. In basso il rate limits di alcune funzioni messe a disposizione dalle api di twitter. La seconda colonna è quella che ci interessa perchè è associata all'app di rete creata sul dev *twitter.com*. In particolare rappresenta il numero di richieste nell'arco di 15 minuti. In questa fase del progetto, è stata utilizzata la user_timeline (1500 richieste in 15 minuti).

GET statuses/lookup	statuses	900	300
GET statuses/mentions_timeline	statuses	75	0
GET statuses/retweeters/ids	statuses	75	300
GET statuses/retweets_of_me	statuses	75	0
GET statuses/retweets/:id	statuses	75	300
GET statuses/show/:id	statuses	900	900
GET statuses/user_timeline	statuses	900	1500
GET trends/available	trends	75	75
GET trends/closest	trends	75	75
GET trends/place	trends	75	75
GET users/lookup	users	900	300
GET users/search	users	900	0
GET users/show	users	900	900

Cleaning

Simboli,punteggiatura,caratteri speciali e parole molto frequenti ma con scarso contenuto informativo (da un punto di vista statistico), sono molto frequenti in qualunque lingua parlata. La funzione clean_text elimina queste parole per migliorare i risultati. In particolare l'operazione di filtraggio dei termini avviene in primo luogo mediante il loro riconosciamento per mezzo

delle espressioni regolari, che rappresentano l'elemento chiave della fase di cattura, in secondo luogo da una pulizia di caratteri speciali e di punteggiatura. Ultima fase della funzione,è quella di appogiarsi ad un file creato opportunamente, contenente molti termini della grammatica inglese di uso comune e qualche abbreviazione. In tal modo si garantisce un esito migliore nelle fasi successive del progetto, in particolar modo nella classificazione e nel calcolo dello schema di ponderizzazione tf-idf per la tematicha Similarità.

```
If not re.search('https?|RT|[^A-Za-z]|amp|[ah|ha]+', list_splitted[i][j]):
list_splitted[i][j]=re.sub('•|‘|"|”|!|“|,|:|&|;|/|\+|\?|...|[.]+|-|–|—|→|\(|\)', '', list_splitted[i][j].lower())
        if not (len(list_splitted[i][j]) < 4 ):
            if not (any(list_splitted[i][j].lower() in s for s in grammarlist)):
                list_clean.append(list_splitted[i][j].lower())
```

Jaccard e modello vettoriale

Il coefficiente di jaccard non considera la frequenza dei termini, in particolare termini rari in un documento contengono più contenuto informativo rispetto un termine molto frequente, jaccard non considera questa evenienza. Formula.

$$\frac{|A \cap B|}{|A|+|B|-|A \cap B|}$$

Il modello vettoriale

Viene usato uno schema di ponderizzazione dei termini di ogni documento (TF-IDF) e sulla base di questi vettori si calcola la funzione coseno. La pesatura dei termini è stata calcolata dalla seguente formula:

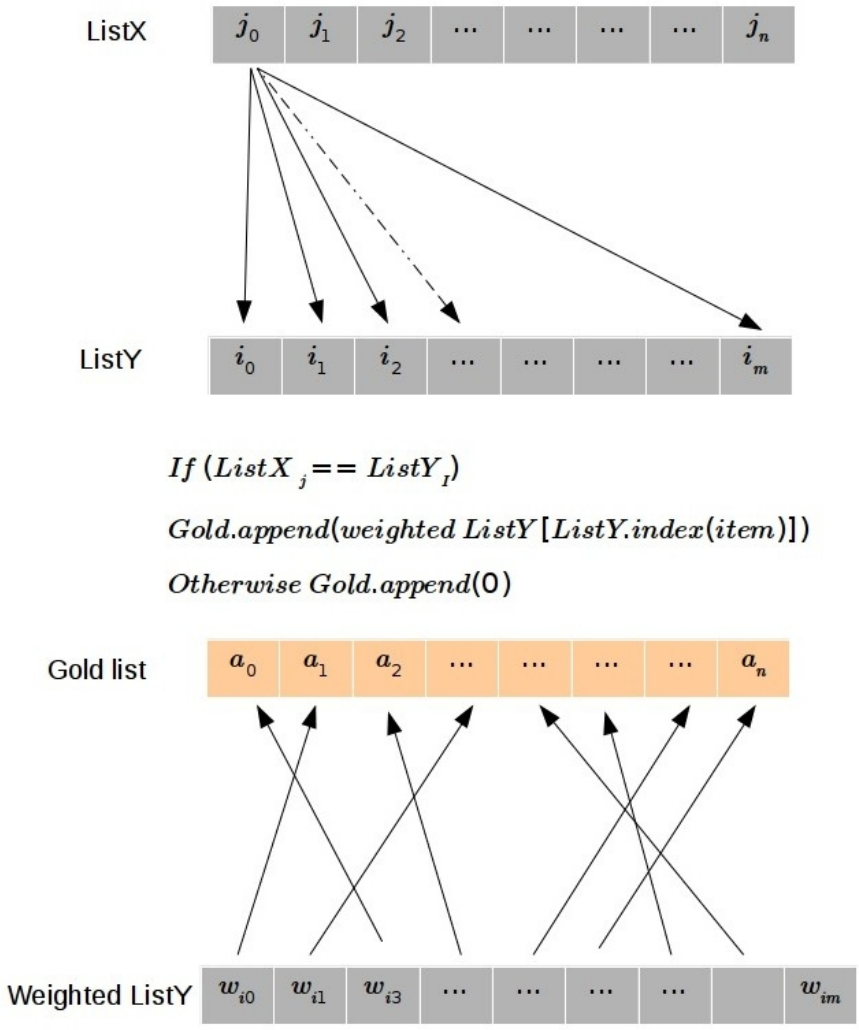
$$w_{i,j} = 1 + log\, f_{i,j} * log\left(1 + \frac{N}{n_i}\right)$$

I termini ponderati migliorano la qualità dell'insieme delle risposte. Per caratterizzare l'importanza del peso si associa $W_{ij} > 0$ ad ogni termine k_i del doc_j. Se k_i non c'è in d_j allora $W_{ij} = 0$. Infine si adotta una normalizzazione della lunghezza del documento(in un contesto generale, documenti più lunghi hanno maggiore probabilità di essere recuperati per via delle maggiore frequenza dei termini, per compensare quest'effetto si usa la sottostante equazione).

$$Similarity(j_0 j_1) = \frac{\sum_{i=1}^{t} w_{i,j0} * w_{i,j1}}{\sqrt{\sum_{i=1}^{t} w_{i,j0}^2} * \sqrt{\sum_{i=1}^{t} w_{i,j1}^2}}$$

La funzione sim_vectorial riproduce la funzione coseno del modello vettoriale ed è riportata in basso.

```
def sim_vectorial(doc_w1,doc_align_2,doc_w2):
    scalar_product=0.0
    lenght_norm2=0.0
    lenght_norm1=0.0
    for i in zip(doc_w1,doc_align_2):
        scalar_product= scalar_product + i[0]*i[1]

    for i in doc_w1:
        lenght_norm1=lenght_norm1 + math.pow(i,2)

    for a in doc_w2:
        lenght_norm2 = lenght_norm2 + math.pow(a,2)



    if scalar_product == 0: return 0 #totally different
    else:
                                return     (scalar_product/((math.sqrt(lenght_norm1))     *
(math.sqrt(lenght_norm2))))
```



In alto, una schematizzazione del processo di allineamento dei termini dei due documenti, con la relativa assegnazione del peso. Quest'operazione è importante per eseguire il corretto prodotto scalare tra I due documenti (ListX,Gold list).

Modello con wordnet

La funzione tf-idf all'interno del modulo è il cuore dell'algoritmo, si basa sullo schema TF-IDF e fa uso di diverse funzioni descritte in dettaglio in basso. la funziona sim_vectorial è identica a quella usata nel modello vettoriale (descritta in precedenza).

la funzione scan mi permette di stabilire se per il termine j-esimo del doc è già stato calcolato il suo peso (mediante schema di ponderizzazione tf-idf), inoltre, l'algoritmo considera anche i possibili suoi sinonimi per una migliore ricerca ottenendo come si evince dai risultati un miglioramento nella similarità tra il modello vettoriale classico e l'estensione basata su wordnet. Codice in basso:

```
def scan(j,repl_list):
    if not repl_list: return True

    for word2 in repl_list:
            if word2 == j:  return False

    for word2 in repl_list:
        wordFromList1 = wordnet.synsets(j)
        if wordFromList1:
            wordFromList2 = wordnet.synsets(word2)
            if wordFromList2:
                s = wordFromList1[0].wup_similarity(wordFromList2[0])
                if s != None and s > 0.90:
                    return False
    return True
```

Un importante operazione da considerare è quella dettata dalla riga 147 alla 166 al fine di allineare correttamente i pesi dei termini del secondo doc rispetto al primo.Questo procedimento è fondamentale al fine di calcolare correttamente il prodotto scalare dei termini dei due doc. Una similarità molto elevata tra un termine i-esimo dal doc_1 con un termine j-esimo del doc_2 indica che la parola è identica oppure che è un suo sinonimo. (figura)

```
for word1 in not_replicated[0]:
    t=0.
    flag=False
    if word1 in not_replicated[1]:
        gold.append(doc_mix[1][not_replicated[1].index(word1)])
        flag=True

    if not flag:
        for word2 in not_replicated[1]:
            wordFromList1 = wordnet.synsets(word1)
            if wordFromList1:
                wordFromList2 = wordnet.synsets(word2)
                if wordFromList2:
                    s = wordFromList1[0].wup_similarity(wordFromList2[0])
                    if s != None and s > t:
                        t = s
                        save_for_index = word2

        if (t > 0.90):
            gold.append(doc_mix[1][not_replicated[1].index(save_for_index)])
        else:
            gold.append(0.0)
```

sim_word-net viene usato per il calcolo dell'inverse document frequency (idf) necessario per unificare la formula utilizzata per il calcolo del peso del termine nel documento considerato (in tal modo si aumenta il peso dei termini rari). Si scansiona l'altro doc al fine di trovare un grado di similarità tra quest'ultimo e il termine in esame, se vi è un alto grado di sim > 0.9 allora è un sinonimo, questo si traduce con la presenza dell'elemento nel doc (incrementando la variabile nella formula per il calcolo idf).

```
def sim_word_net(j, z1):
    wordFromList1 = wordnet.synsets(j)

    for word2 in z1:
        if word2 == j: return True
    for word2 in z1:
            wordFromList1 = wordnet.synsets(j)
            if wordFromList1:
                wordFromList2 = wordnet.synsets(word2)
                if wordFromList2:
                    s = wordFromList1[0].wup_similarity(wordFromList2[0])
                    if s != None and s > 0.90:
                        return True
    return False
```

Risultati

In basso la tabella comparativa tra coppie di utenti con i relativi tempi di calcolo in secondi ed i modelli utilizzati. Un imporante risultato ipotizzato e dopo dimostrato , come si evince dai valori è il seguente.
Data una coppia di utenti ed il loro coefficiente di similarità del modello vettoriale e del modello vettoriale supportato da wordnet, I valori affermano che:

$$\text{Sim}_{\text{vector\_model}} \leq \text{Sim}_{\text{vector\_model\_with\_wordnet}}$$

La disequazione è del tutto lecita. Il modello con wordnet considera anche I sinonimi per un dato termine.

Comparison of the results

	Number posts	Vector model	with wordnetV2	Jaccard coefficient
Trump Clinton H.	800	**0.2632**	**0.3083**	0.08474
Time		0.20s	559s	0,0003s
Osho DalaiLama	800	**0.1368**	**0.1577**	0.0252
Time		0.23s	300s	0,0004s
Tyson N. Van Gogh	800	**0.0864**	**0.117**	0.0457
Time		0.25s	510s	0,0002s
Tyson N. Michio Kaku	800	**0.1454**	**0.19**	0.0583
Time		0.29s	703s	0,0003s

Conclusioni

Una debolezza del modello vettoriale nella forma generale, è che è basato esclusivamente sulla frequenza dei termini relativi, di fatto costituisce un problema nell'attribuire una similarità affidabile a documenti che trattano lo stesso argomento ma espresso con sinonimi. Ad esempio "La serie tv mr-robot è molto amata dai programmatori" "I sviluppatori software adorano il telefilm mr robot" . Questo semplice esempio mette in chiaro una debolezza del modello vettoriale generale.

Nella versione 2.0, Il software utilizza word net, per associare un certo grado di similarità ad ogni termine del doc. In questo modo (rispetto al modello vettoriale generale) si ha un grado di accuratezza leggermente maggiore, di contro, naturalmente, la complessità computazionale aumenta notevolmente,queste per via del ritrovamento di possibili sinonimi sia per quanto riguarda la costruzione dello schema di ponderizzazione tf - idf, sia per quanto riguarda la fase finale (ossia quella in cui bisogna relazionare il term i-esimo del doc 1 con il term j-esimo del doc 2 con similarità molto alta(sinonimo) in modo da effettuare il corretto prodotto scalare al fine di valure la similarità mediante il vector model).

I risultati finali dei modelli vengono comparati con il rispettivo grado di sim ed il loro tempo di esecuzione ed è prevista la possibilità di salvarli in un file di testo.

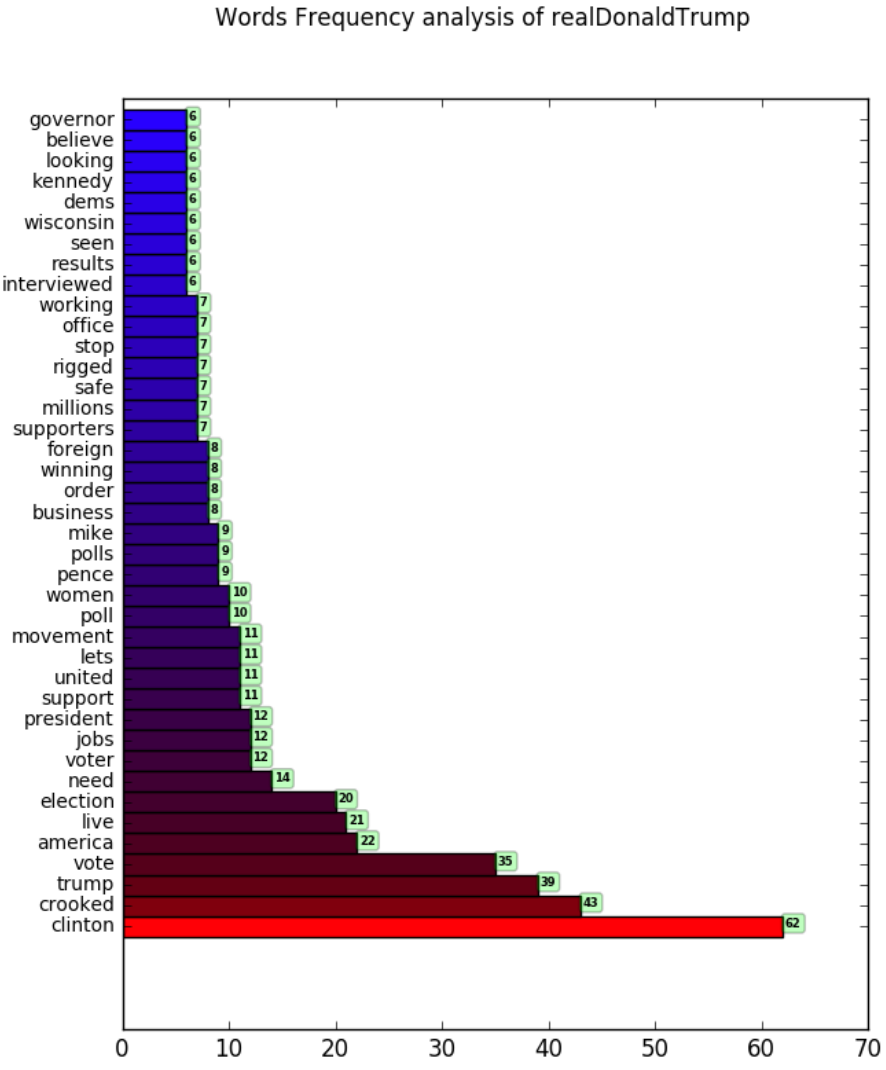
(*La versione 1.0 con wordnet è rimasta provvisoria, sulla base di questa è stato definito un grado di accuratezza e completezza maggiore in modo da concretizzarlo nella versione 2.0*)

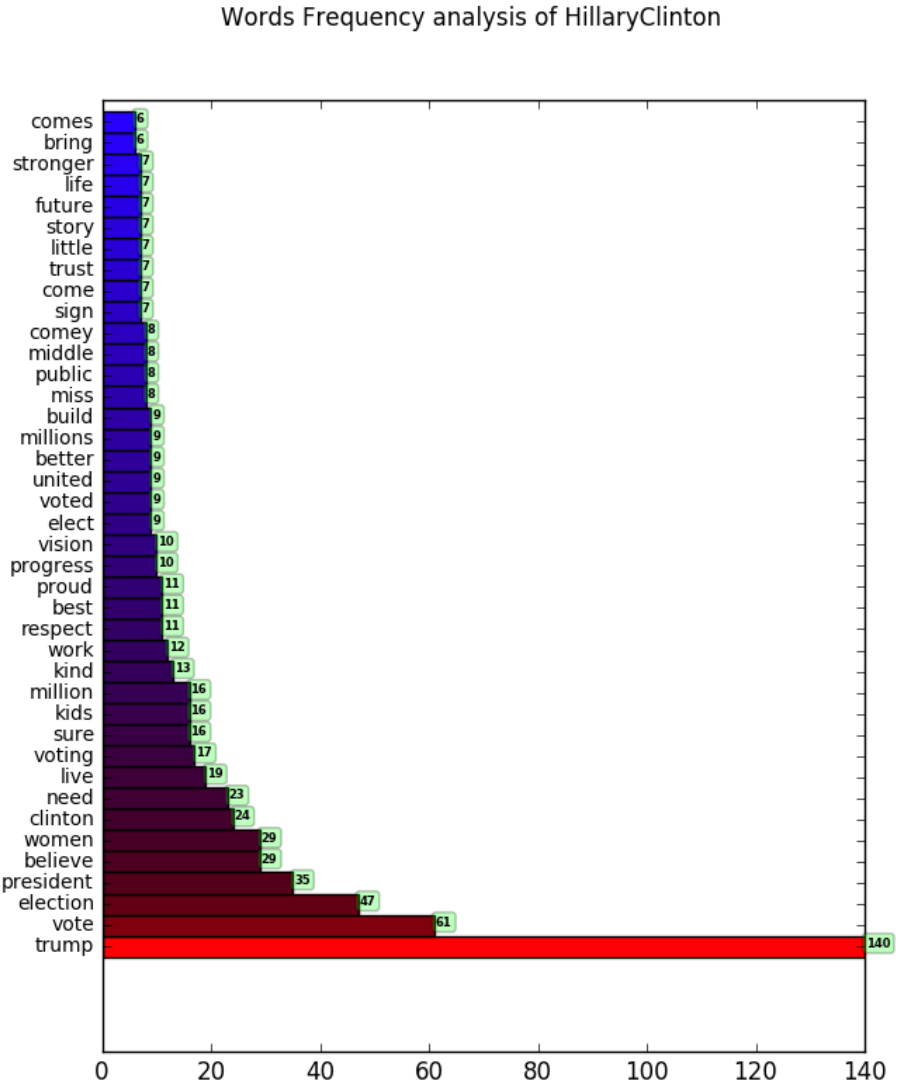
III. Analisi dei risultati

Durante la fase di testing sono stati analizzati in dettaglio gli utenti Trump ed Hillary Clinton. Il file di testo generato dal software è descritto in basso.

area di plotting:

Per una migliore analisi visiva, ho optato per l'utilizzo delle librerie matplot. In particolare la funzione analysis_word, crea un grafico a barre orizzontali, che mostra la frequenza dei termini più utilizzati dai due target.

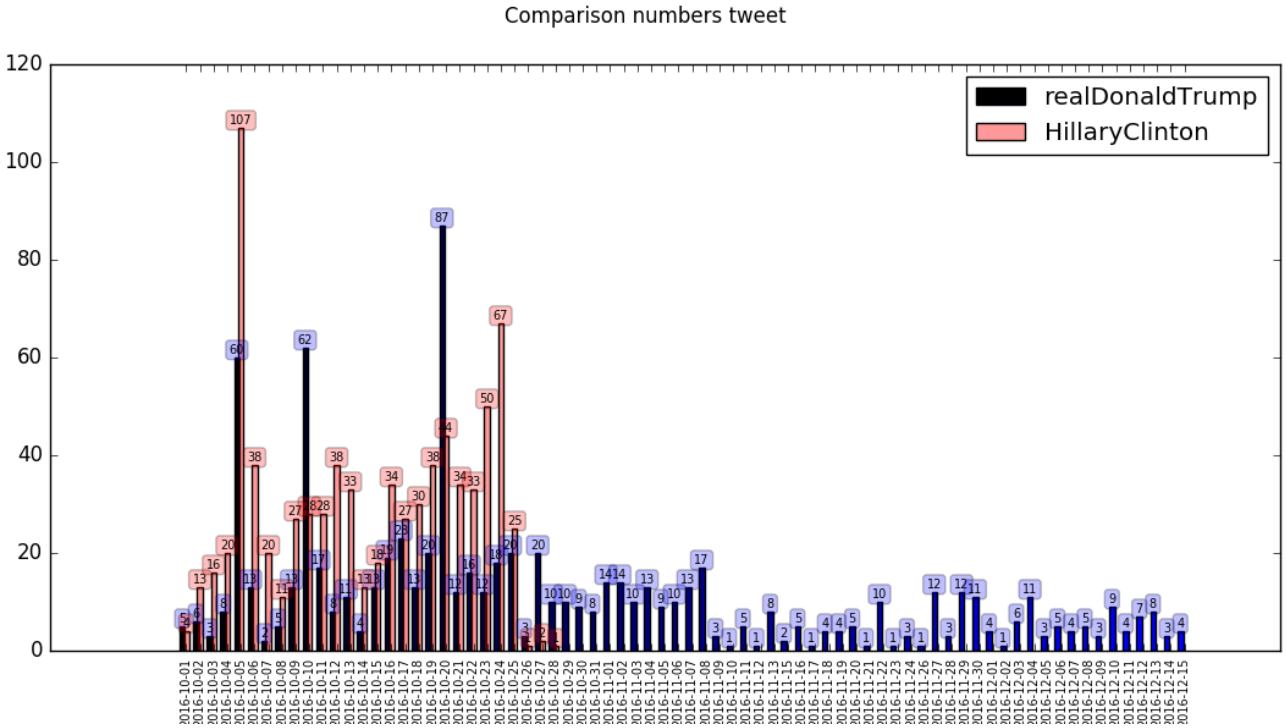




(da notare la variabile my_colors1 per attribuire un colore più caldo alle parole ad alta frequenza)

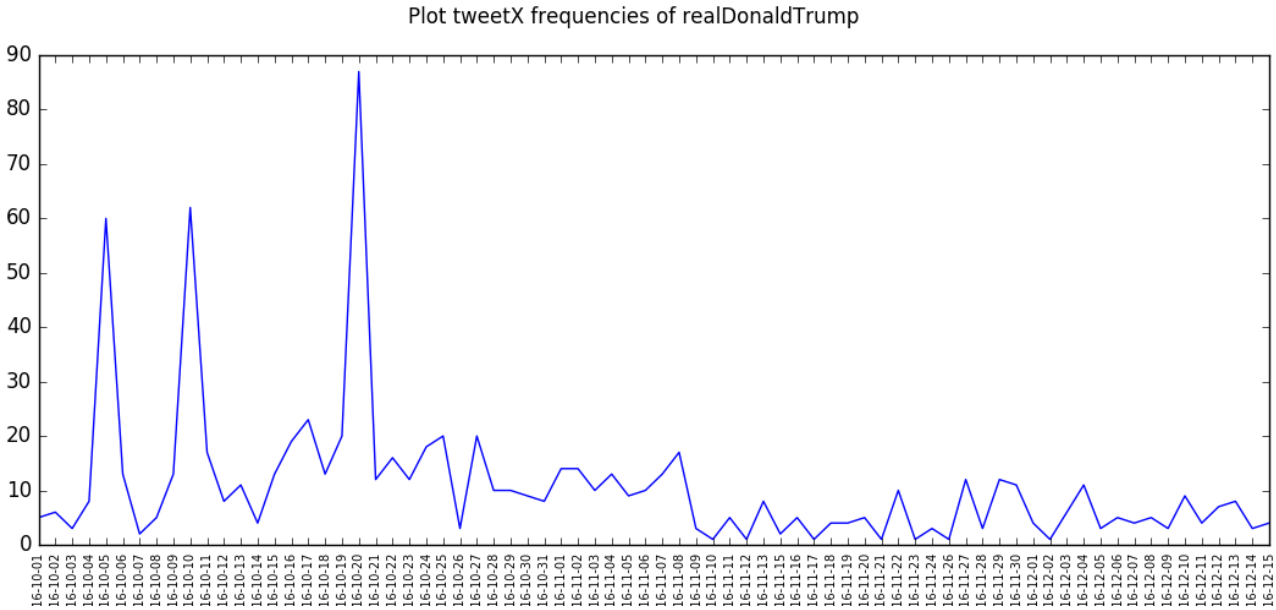
```
my_colors1 = [(1.0 / (1.0 + np.math.log2(x)), 0.0, x / len(w)) for x in range(1, len(w) + 1)]
```

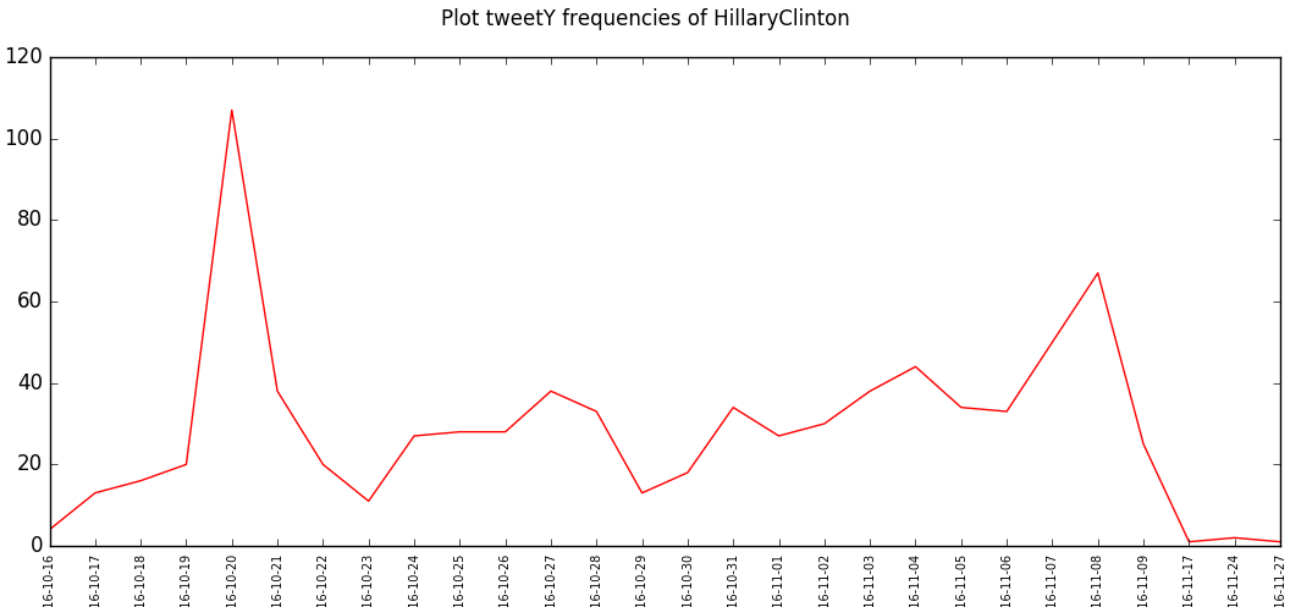
La funzione analysis_frequency crea un grafico unificato tra il numero di post giornalieri dei due target e li mette a confronto. Sull' ascisse troviamo le date dei post e sull'ordinata la frequenza dei post.



Studi mirati sui particolari target mettono in evidenza alcune caratteristiche. Durante la fase di test ho esaminato, tra i vari target osservati, due di por-

tanza maggiore, che durante le fase della candidatura per la casa bianca sono emersi maggiormente rispetto gli altri. Dai grafici in basso si evince che il numero di entrambi i candidati trova il picco nelle fasi finali delle votazioni , per poi descrescere notevolemente.La candidata Hillary Clinton ha un picco di discesa molto rapido dopo gli esiti. Trump mantiente un media sul numero di tweet giornalieri notevole. Naturalmente si possono osservare target di diversa natura, il software è generalizzato e non specializzato solo su questioni politiche. La funzione termina generando un semplice plot dei due target (sulla base degli stessi dati).
Nella raccolta dei dati, ho analizzato 3235 tweet di trump e 3235 tweet di clinton nelle fasi pre/post elezioni. In particolari è stato estratto da ogni post il datatime e raggrupato per mese nel grafico finale. Si evince un netta curva in aumenta nelle fasi delle votazioni per entrambi i candidati (trump mantiene una media oltretutto, la clinton è scomparsa dopo le elezioni)




Plot tweetY frequencies of HillaryClinton
120
100
80
60
40
20
0
16-10-16
16-10-17
16-10-18
16-10-19
16-10-20
16-10-21
16-10-22
16-10-23
16-10-24
16-10-25
16-10-26
16-10-27
16-10-28
16-10-29
16-10-30
16-10-31
16-11-01
16-11-02
16-11-03
16-11-04
16-11-05
16-11-06
16-11-07
16-11-08
16-11-09
16-11-17
16-11-24
16-11-27

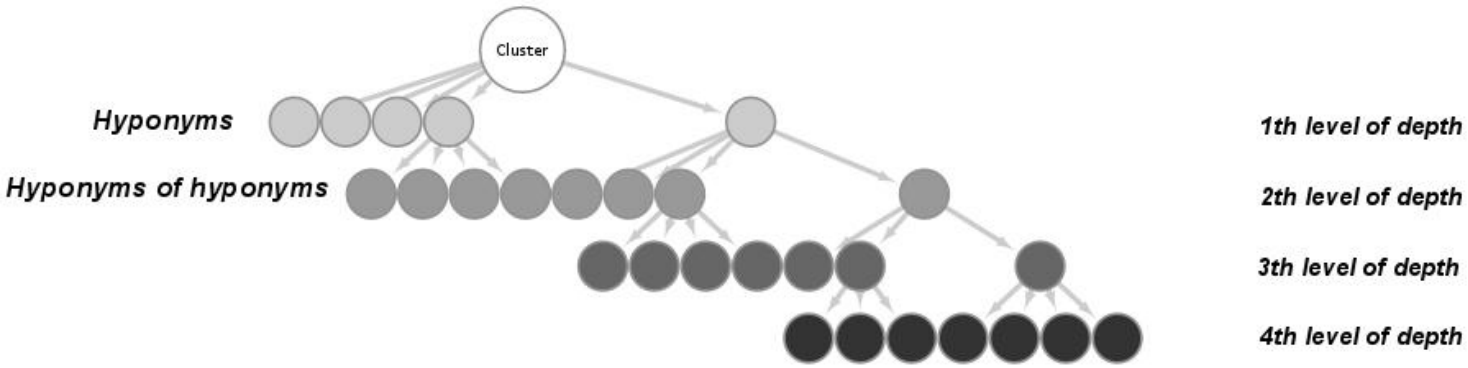
Approcci alla Classificazione (versione 1.0 e 2.0)

Introduzione

Obiettivo supplementare finale (ancora da migliorare) è stato quello di realizzare un classificatore 'naive' (ed uno un pò meno) che ha come obiettivo l'assegnazione di una label ad un utente in modo da estrarre possibili caratteristiche sugli interessi generali del target considerato. Gli scopi sono molteplici: Suggerire agli utenti la possibilità di seguirne un altro (che ha interessi comuni), migliorare l'esperienza dell'utente suggerendo notizie sulla base dei propri interessi. Un utente che parla frequentemente di astronomia, troverà con piacere possibili notizie su quest'argomento o suggerimenti di astrofisici da seguire su twitter come possibilità.
Ci tengo a ricordare che entrambe le versioni possono essere viste come "Approcci" iniziali alla classificazione di natura molto statica.

Classificatore versione 1.0 *(Classificazione gerarchica mediante una struttura ad albero di istanze appartenenti ad un dato gruppo di concetti)*

In questa versione (molto naive) non si tiene in considerazione alcuna statisticha legata alla frequenza delle parole, ma piuttosto , alla presenza o all'assenza di un termine del documento da classificare con il relativo albero di iponimi (appartenenti al cluster). Il sistema crea un albero di iponimi per ogni categoria (fisica,filosofia,astronomia...). Ogni albero di iponimi si estende fino ad una profondità di 4° livello, in definitiva si considerano gli iponimi degli iponimi degli ..... etc. In tal modo si spera di trovare una parola specializzante nell'albero i-esimo e di conseguenza una relazione col termine j-esimo del documento da etichettare.

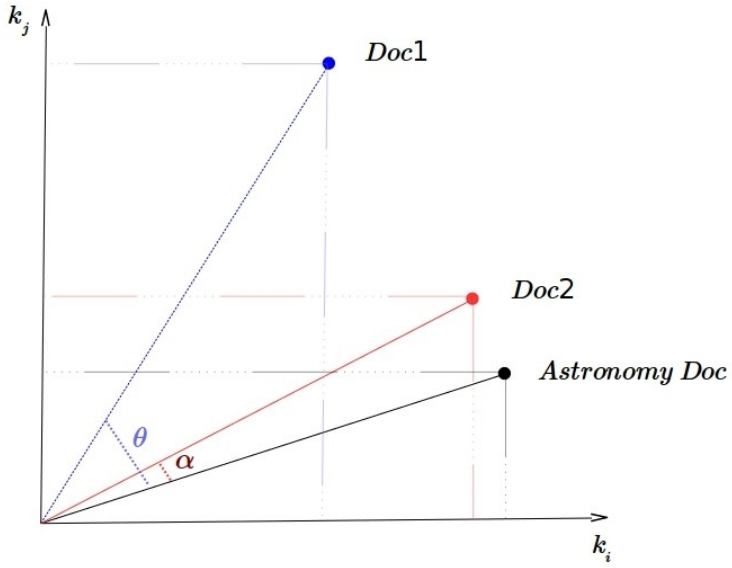


Naturalmente si tengono in considerazione non tutti i termini del documento non etichettato, ma soltanto i primi n più frequenti. Un utente che utilizza frequentemente la parola 'mantra', molto probabilmente tratterà argomenti riguardanti la filosofia orientale.
Debolezza del sistema: non vi è alcuna distinzione tra un termine molto frequente ed uno meno frequente, i termini sono trattati allo stesso modo.

Classificatore versione 2.0

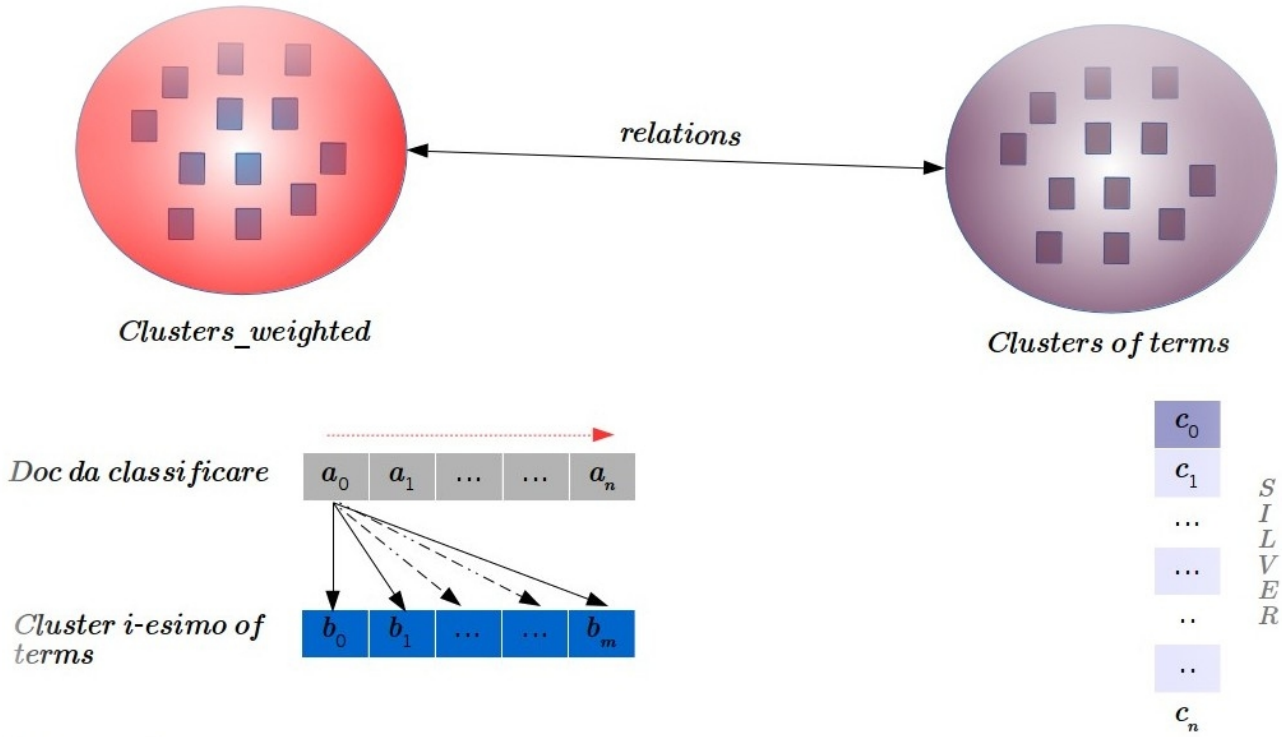
Il classificatore nella versione 2.0 risulta essere più potente del precedente perchè non considera semplicemnte la presenza o l'assenza di quel termine nell'albero di istanze appartenenti ad un dato gruppo di concetti (scienza,arte,filosofia...). Utilizza lo schema di pesatura TF-IDF per la ponderizzazione dei termini del documento e del set di clusters (documenti etichettati da un essere umano, presi dal sito www.gutenber.net, approccio

supervisionato). La similarità tra un documento da etichettare ed ogni cluster(documento) viene calcolata mediante il modello vettoriale. (la versione 2.0 ingloba parte della versione 1.0 per motivi di programmazione). Nella figura in basso una rappresentazione minimale della funzione coseno per il calcolo della similarità.



Si nota che il doc2 ha un valore molto più alto (in termini di coseno) rispetto al doc 1 verso il doc di astronomia, il chè fornisce un suggerimento sull'appartenenza del doc2 alle tematiche attinenti all'astronomia. La rappresentazione in figura è bidimensionale.(un documento è rappresentato da un vettore di termini pesati con t-dimensioni).

Schematizzazione della versione 2.0:




```
If ( a == b )
silver.append(clusters_weighted[i][clusters_of_terms[i].index(clusters_of_terms[i][j])])
```

Test

I risultati dei test sono schematizzati nella tabella in basso e dimostrano che il classifierV2 nella maggior parte dei casi ha una precisione maggiore rispetto la versione basata sugli iponimi.

	ClassifierV1	ClassifierV2	
Users	Category		
Osho	music	Orientalism 0.1096	Politics 0.1014
totokokolabel	music	Music 0.0713	Politics 0.0416
PhilaOrchestra	music	Art 0.1365	Music 0.1361
OlafurArnalds		Orientalism 0.1647	Music 0.1611
yoshikiofficial	music	Art 0.1106	Politics 0.0971
neiltyson		Physics 0.2268	Astronomy 0.2224
BadAstronomer		Astronomy 0.1929	Art 0.1832
umutayildiz		Physics 0.1284	Astronomy 0.1233
KirkDBorne	music art	Music 0.1643	Politics 0.1638
AstroKatie		Astronomy 0.2096	Politics 0.1983
CatherineQ	music	Astronomy 0.2034	Orientalism 0.2029
DeviantArt	art	Art 0.1799	Astronomy 0.1862
artsy	phylosophy	Art 0.1677	
ArtFagCity	music history	Music 0.004	Orientalism 0.1534
Buddhism_Now	music	Politics 0.1428	Philosophy 0.1414
DalaiLama	music	Politics 0.1750	Art 0.1394
LamaSuryaDas	music	Orientalism 0.1806	Politics 0.1791
GreggDCaruso		Politics 0.1999	Orientalism 0.1949
philosophybites	music	Orientalism 0.2049	Philosophy 0.1974
SethShostak	physics music	Politics 0.1073	Music 0.1050
GuyLongworth		Orientalism 0.2258	Politics 0.2214
vangoghmuseum		Art 0.1460	Astronomy 0.1335

Miglioramenti futuri

- Ho notato che, qualora dovessero presentarsi termini composti da due parole in wordnet, il sistema include un '_' al posto dello spazio. Il software, attualmente, non gestisce l'eccezione.
- Migliore è la funzione per la pulizia del testo originale, migliore sarà il risultato finale. Molte delle parole inutili vengono scartate, simboli particolari, termini troppo corti, link, tag, molti termini grammaticali contenuti nel file creato grammar_file.text, etc... Al fine di aumentare la precisione del classificatore bisognerebbe escludere anche le parole che non sono 'specializzanti', ad esempio termini come 'home','go','glass' non trovano posto in alcuna categoria considerata nel software. Invece , parole come 'planet','energy','universe','bhagavad_gita' hanno successo per via delle loro caratteristiche specializzanti rispetto alle categorie considerate, è inevitabile trovare le prime tre come iponimi di scienza e l'ultima come iponimo di filosofia orientale (questo per la versione 1.0)
- La versione 2.0 somiglia molto al Knn, anche se quest'ultima, nella versione classica prende in considerazione la retta di minima distanza tra un documento con un cluster per poi inglobarlo e ricalcolare i centri(la distanza euclidea è una cattiva idea perchè doc molto simili ma con lunghezze ben diverse avranno distanza molto elevata).Qui si utilizza la funzione coseno per la similarità ed attualmente non è previsto alcun miglioramento del cluster(nella versione successiva sarà introdotto)
- Il software non tiene conto di coppie di termini visti come unico concetto, ad esempio intelligence artificial verrà visto come due termini (intelligence, artificial). Migliorabile.

Conclusioni 2°parte

Nella versione molto naive:
Il software migliora con il perfezionamento della tassonomia di una determinata categoria. Nello specifico, con l'estensione di iponimi di una albero di concetti,aumenta la precisione del classificatore. Quindi estendendo wordnet con più termini, il software migliorerà notevolmente.
Nella versione meno naive: Si comporta abbastanza bene con qualunque tipologia di documento.
In definitiva , entrambe le versioni, come dimostrano i dati, trovano dei risultati gradevoli.I concetti dietro ad entrambe le versioni sono molto semplici, ma la gestione di indici vari in fase di programmazione è leggermente impegnativa.