Twitter Acquisition

Progetto per il corso di Sistemi Intelligenti per Internet, Università degli Studi di Roma Tre

Realizzato da: Martina Mangione

Basato su: Online Adaptive Topic Focused Tweet Acquisition di M.Sadri, S. Mehrotra, Y.Yu

Versione: 2.0

Si ringrazia Mehdi Sadri per il supporto e per aver acconsentito alle modifiche del codice
https://github.com/mehdiesadri/tweet-acquisition

RIEPILOGO ESECUTIVO

Il progetto originale

Twitter fornisce un'API di streaming pubblica strettamente limitata, che rende difficile ottenere contemporaneamente una buona copertura e rilevanza quando si monitorano i tweet per uno specifico argomento di interesse. Mehdi Sadri, Sharad Mehrotra e Yaming Yu hanno proposto in una loro pubblicazione del 2016 (citata nella prima pagina di questa documentazione) un sistema di acquisizione di tweet per migliorare il loro monitoraggio, in base alle esigenze di un client in modo che la qualità e la quantità dei risultati migliorino nel tempo.

Obiettivi

L'obiettivo che mi sono posta con questo progetto è di analizzare l'algoritmo e di mostrarne il funzionamento e l'efficacia attraverso degli esempi. Il progetto è stato rivisitato, eliminando librerie obsolete non più disponibili e migliorato aggiungendo dei log che segnalano all'utente quali operazioni sono in atto. Il lavoro svolto è stato visionato da M. Sadri che ha acconsentito alle modifiche della Java Application da lui realizzata.

Le modifiche

Il nuovo progetto a differenza del precedente è sprovvisto di Dimple, una libreria Javascript che consentiva di recuperare da una base di conoscenza esterna i termini più comuni all'interno di fonti di dati esterne come notizie. Queste nuove informazioni sarebbero servite ad arricchire il set di frasi che compongono l'Interesse (ossia l'oggetto Java "Interest") per mezzo delle quali l'applicazione avrebbe recuperato i tweet più rilevanti. Consente inoltre un ordinamento, in base alla rilevanza, dei tweet.
E' stato quindi eliminato dal progetto il package topK e sono state aggiornate tutte le dipendenze del POM.
Perché si è scelto di eliminare Dimple? Il webJar presente nel Maven Repository non è più funzionante in nessuna delle versioni in cui viene fornito. Secondo M.Sadri per la parte presa in considerazione per questo progetto non è necessario: migliorerebbe i risultati certo, ma il sistema funziona ugualmente.
Per ampliare l'acquisizione dei dati, l'applicazione inoltre verifica se i termini presenti all'interno del tweet siano tra i più comuni collegandosi su un server (sensoria.ics.uci.edu:9090). Non disponendo dell'autorizzazione a questo server, è stato deciso di impiegare un dump di wikipedia su cui è stato lanciato il seguente progetto:
https://github.com/marcocor/wikipedia-idf
Ciò ha permesso di recuperare per ogni parola all'interno di Wikipedia la sua frequenza e la Inverse Document Frequency. Quest'ultimo indicatore misura la frequenza inversa di una parola tra tutti i documenti. E' molto alto nei termini specifici, mentre è molto basso nelle parole comuni.
Il dump che è stato utilizzato per i test a seguire è consultabile al seguente link:
https://dumps.wikimedia.org/enwiki/20190820/
Il file impiegato è enwiki-20190820-pages-articles.xml.bz2 (15.3 GB)
Nel caso il termine non sia presente nel file csv generato, si stabilisce un punteggio di default per ogni frase che è 0.5(stabilito da Sadri stesso in mancanza del collegamento al server)
La dipendenza esterna al secondo progetto di M.Sadri per il Natural Language Processing è stata eliminata e le classi che lo componevano sono state inserite direttamente nel progetto twitter-acquisition.

Per il riconoscimento dei Tweet in lingua italiana è stato aggiunto nel package “lang” la classe ItalianClassifier che confronta la parola passata con quelle presenti in un dizionario collocato nella cartella src/main/resources/profiles/it. Le modifiche sono inoltre visibili consultando la history del progetto git.

Struttura del progetto

Il progetto è suddiviso in due parti:

1. **Natural Language Processing:** Si occupa di riconoscere la lingua con cui sono scritti i tweet.
   I package che si occupano dell’operazione sono:
   - package **txt:** normalizza il testo eliminando le stopping word(classe StopWordDetector e maiuscole (classe TextNormalizer) e avverbi e preposizioni che non sono utili alla ricerca dei tweet e aumentano i tempi di risposta del sistema.All’interno della cartella src/main/resources sono presenti: la cartella profiles che contiene diversi dizionari di parole in diverse lingue, il file MASTER che corrisponde al dizionario della lingua inglese e stopwords che colleziona gli avverbi e le proposizioni che si intende eliminare dai tweet sia in inglese che in italiano. In caso di variazioni morfologiche delle parole non sono stati utilizzati stemmer.
   - package **lang:** contiene le classi EnglishClassifier e ItalianClassifier che si occupano di classificare e riconoscere la lingua
2. **Tweet Acquisition:** si occupa dell’acquisizione dei Tweet e del calcolo della rilevanza delle frasi per ogni Interest inserito dall’utente. Verrà dato maggiore spazio alle operazioni che vengono eseguite nel prossimo capitolo

Per il suo funzionamento, il progetto utilizza **MongoDB** , un DMBS non relazionale basato su documenti , che permette di effettuare ricerche full-text. Questo è molto importante perché consente di mantenere in memoria gli Interest e di memorizzare i report di ogni esecuzione. Il database “dataworld” si occuperà quindi di contenere le entità Client, Interest, Location, Phrase, Report, Query, Tweet e User che grazie al driver Morphia verranno mappate con le classi Java presenti nel package **conf.** Morphia, come si può’ intuire, è quindi un ODM (Object Document Mapper) che ha il compito, di effettuare il mapping delle classi Java con le entità del database, come JDBC per i database relazionali.
La configurazione di MongoDB è modificabile nel file **config.txt.**
Questo file consente inoltre di inserire le API Twitter, tracciare le risorse che ci servono e configurare la lingua. Al momento è impostato l’inglese come si può’ vedere dalla riga 54: `LanguageCheckLanguage=en`
Il package conf, tuttavia, non contiene solo le entità del database Mongo ma ospita anche la classe ConfigMgr e l’InterestStoreManager. La prima si occupa di leggere i parametri dal file config.txt la seconda invece permette, se eseguita al posto del main, di inserire, rimuovere e i mostrare gli Interest.

INSTALLAZIONE ED ESECUZIONE

Java Application

Il progetto è disponibile al seguente link: https://github.com/MARMANGIONE/SIITwitterProject
Può essere importato in qualsiasi Java IDE. Nel caso di Eclipse basterà andare su File —> Import —> Project from Git e, una volta cliccato su Clone URI, incollare nell'apposito campo di test il link del repository git.
Importato il progetto nel proprio workspace occorrerà installare il JAR e le sue dipendenze e per farlo si deve cliccare con il tasto destro sul file pom.xml o sulla cartella del progetto sul comando **"Run as"** e poi **Maven Install**. Installato l'artifact, prima dell'esecuzione del progetto, occorrerà avviare da terminale il proprio database in locale (si consulti per tale scopo il secondo paragrafo di questo capitolo).
Avviato il database, la Java Application può' essere eseguita in due modi:
1. **InterestStorageManager** permette di stabilire una connessione con il database Mongo e consente di aggiungere, visualizzare e rimuovere gli Interest. L'interest è caratterizzato da un id, un nome, un client-id, un client-name e da una lista di frasi che sono quelle che ci permetteranno di effettuare la ricerca.
Come primo approccio si è deciso di utilizzare l'esempio presente nel paper.

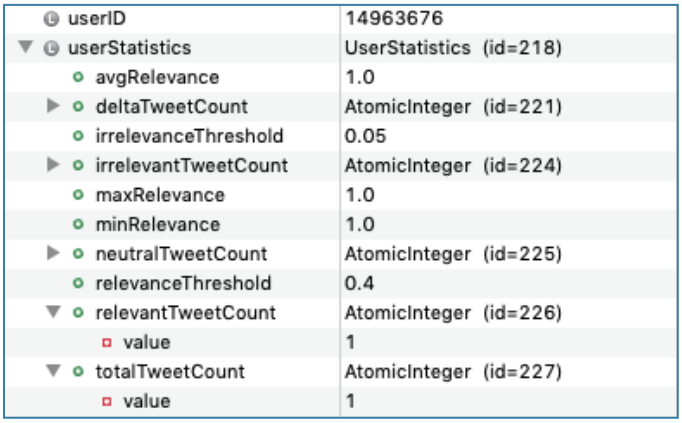
```
- In the Interest Manager you can add or remove different Interests to acquire relevant tweets.
 Type "help" if you want to know the commands to perform these operations
help
show,add, rm iid, user, exit
Type the command you want
add
At the moment you can add interests and phrases only in English
- Enter ClientId:
1
- Enter ClientName:
Martina
- Enter interestId:
1
- Enter interest topic name:
US Presidential Debate
- enter phrases one by one or a file path containing phrases one per line:
presidential debate
republicans debate
democrates debate
clinton drought
```

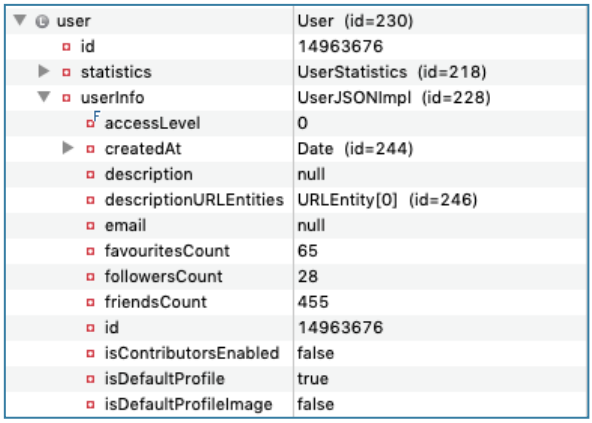
Come si può notare, le frasi sono semplici e piuttosto sintetiche. La ricerca infatti non viene fatta per aggregazioni di termini ma per singole parole.Quando le frasi diventano articolate l'algoritmo perde di efficacia, questo perché il natural language processing non fa correttamente il proprio lavoro. Per questo si potrebbe pensare di sostituire la libreria creata da Sadri, che si basa su un dizionario di termini, con la Standford NLP (https://github.com/stanfordnlp/CoreNLP).
L'interest, una volta definito, è quindi inserito nel database ed è pronto per essere selezionato in qualsiasi momento

2. **MainTA** permette in base all'interesse scelto di iniziare l'acquisizione dei Tweet. Quando viene eseguito vengono avviati diversi thread: uno per l'avvio dello stream di Twitter, il secondo per l'acquisizione in base ai filtri (ossia le frasi inserite precedentemente nell'Interest). Mentre l'applicazione è in ascolto si possono eseguire molteplici operazioni digitando semplici comandi da tastiera:

```
smc – Storage Manager Queue Size
tc – Total Tweet Count
wtc – Window Tweet Count
tce – Total English Tweet Count
tcr – Total Relevant Tweet Count
wtcr – Window Relevant Tweet Count
tcd – Total Delta Tweet Count
tcp – Total Processing Buffer Tweet Count
```

Lo storage Manager si occupa di inizializzare il database e di creare una coda con i Tweet che poi vengono memorizzati nell'Interest scelto (non vengono persistiti nel database). Una volta che i tweet vengono inseriti, essi sono eliminati dalla coda.
Lo storageManager avvia delle statistiche sui tweet appena memorizzati e sui loro autori.
Queste ultime informazioni vengono poi salvate nella collection "user" del database dataworld.
Tutto ciò viene fatto nel metodo `storeTweet(Tweet tweet).`
Nelle immagini in basso si può vedere come il programma gestisce e memorizza i seguenti oggetti:





Il **total tweet count** indica il numero di tweet complessivo che sono stati intercettati dal Listener, rilevanti e non rilevanti, in tutte le esecuzioni. Quindi non tiene traccia solo della sessione corrente ma anche delle precedenti. Il conteggio dei tweet che sono stati recuperati nella sessione corrente è dato invece dal **Window Tweet Count**. Se si vuole una statistica più precisa sui tweet rilevanti occorre inserire da tastiera il comando wtcr. Il T**otal Delta Tweet count** invece tiene traccia dei tweet nuovi che sono stati acquisiti nella sessione corrente. Nell'esempio qui sotto si può vedere una prima esecuzione del programma scegliendo l'interest id precedentemente creato:

```
ago 16, 2019 10:42:44 PM twitter4j.CommonsLoggingLogger info
INFORMAZIONI: Establishing connection.
ago 16, 2019 10:43:03 PM twitter4j.CommonsLoggingLogger info
INFORMAZIONI: Connection established.
ago 16, 2019 10:43:03 PM twitter4j.CommonsLoggingLogger info
INFORMAZIONI: Receiving status stream.
smc
Storage Manager Queue Size: 0
tc
Total Tweet Count: 4
wtc
Window Tweet Count: 5
tc
Total Tweet Count: 5
tcr
Total Relevant Tweet Count: 5
wtcr
Window Relevant Tweet Count: 5
tcd
Total Delta Tweet Count: 0
```

MongoDB

Nel file config.txt alla riga 12 e 13 è presente l'host del database. Nel mio caso il database è in locale.
Per il funzionamento del progetto, in qualsiasi modo esso venga eseguito, è necessario avviare MongoDB.
Questa operazione viene eseguita tramite due finestre del terminale: nella prima digitando `mongod` avvieremo il processo server, nella seconda, invece, digitando `mongo` avvieremo la shell, che ci consentirà di visualizzare il contenuto all'interno del database.
Le frasi che sono state utilizzate per i test presenti in questa documentazione sono recuperabili nel file: `phrases.txt`

TWEET ACQUISITION - IL FUNZIONAMENTO

L'applicazione utilizza le API di Twitter per scaricare tutti i tweet più recenti che corrispondono alla query.
Il programma scarica i tweet e calcola alcune prime statistiche su hashtags, menzioni e autori.
Il processo è iterativo con l'obiettivo di massimizzare l'acquisizione di tweet e di aggiornare l'insieme di frasi che rappresentano la query.
La prima operazione richiesta è quella di definire un interesse (Interest) fornendo un insieme coerente di frasi.

Generazione della query e acquisizione dei tweet

L'obiettivo di questa fase è selezionare le frasi per aumentare il numero di tweet rilevanti raccolti.
Il valore di una frase cambia spesso quindi serve un meccanismo adattivo.
La classe Acquisition attraverso l'oggetto TwitterListener provvede tramite lo stream aperto di TwitterStream(metodo `startListening()`) ad acquisire un insieme di Tweet.
Le frasi presenti negli Interest vengono aggiunte in ordine sparso nei filtri formando l'oggetto FilterQuery, necessario alla libreria Twitter4j.
Per ogni frase si calcola il peso di ogni termine attraverso il metodo computeFrequencies() presente all'interno della classe Interest. Inizialmente è assegnato un peso unitario. Ogni termine viene aggiunto alla lista termFreq. Se questa lista è vuota o non contiene il termine, questo viene aggiunto insieme al suo peso che è pari al rapporto tra la frequenza della frase in cui il termine è contenuto e la frequenza di tutte le frasi associate all'interest(nell'implementazione prende il nome di variabile **d**). La frequenza di un termine altro non è che la somma dei valori d per un determinato termine.
Ad esempio per "debate" si avranno i seguenti valori:

Dal momento che il termine è presente in tre frasi su quattro la sua frequenza sarà pari a 0.75
La frequenza è molto importante perché fornisce una misura comparativa di quanto quella parola sia distintiva per l'interesse corrispondente, e quindi in una fase successiva diventa utile perché consente di costruire delle nuove potenziali frasi da inserire per quell'Interest.
Bisogna infine fare una valutazione sui tweet appena acquisiti.
Un tweet t i cui termini coincidono con tutti i termini di una frase assegnata all'Interest scelto rappresenta perfettamente l'Interest tuttavia non tutti questi tweet sono rilevanti. L'algoritmo infatti considera molto importante la fonte del tweet e il pattern utilizzato. E' essenziale che una frase sia rilevante perché questo è un metro di giudizio per la creazione delle query.

Calcolo della rilevanza

Una frase, facente parte di un Interest, è costituita da un insieme di termini.
Dopo che il TwitterListener ha eseguito l'operazione di Filter, arrivano i tweet e nella classe RelevanceCheck viene effettuato il calcolo della rilevanza.
Nell'oggetto Tweet si eseguono due operazioni:
1. **Il calcolo della rilevanza della frase (getPhraseRelevance())**: per effettuare questa operazione si appoggia a getSatisfaction(). Questo secondo metodo scorre sia i termini presenti nella frase sia i termini presenti nel tweet. Se i termini che compongono la frase e il tweet sono uguali si calcola la frequenza del termine con il metodo getPhraseTermFreq(term). L'oggetto Interest si occupa quindi di effettuare l'operazione associando al termine di ogni frase un valore di frequenza. Il ritrovamento della frequenza è realizzato dal metodo computeFrequencies(). In parole povere, la satisfaction è la somma della frequenza di tutti i termini che corrispondono con i termini presenti nei tweet. Per tenere traccia delle corrispondenze si usa quindi un matchCount. Qui in basso si possono vedere i log che mostrano il calcolo della rilevanza per alcuni tweet che sono stati recuperati:

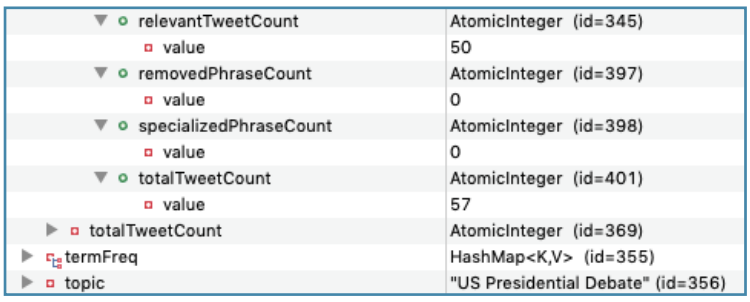
```
rc.RelevanceCheck - 1,00 [1,00      0,00    0,00]       [debate, apompliano, joerogan, retweet, watch, presidential, hosted]
rc.RelevanceCheck - 1,00 [0,00      0,00 ** 1,00]       [pomp, tweet]
rc.RelevanceCheck - 0,13 [0,13      0,00    0,00]       [howie, nod, green, dunleamark, bad, fuel, greennewdeal, greenpeaceusa, hawkin
rc.RelevanceCheck - 1,00 [1,00      0,00    0,00]       [billionaire, race, dfbharvard, snake, samples, better, debate, oil, democrat,
```

Il primo valore a destra indica la rilevanza della frase, il secondo la rilevanza del pattern mentre il terzo la rilevanza dell'utente. Quando un utente può essere considerato una fonte autorevole? Un utente che in passato ha pubblicato tweet pertinenti. Quindi l'importanza dell'utente è data dal valore massimo della rilevanza di tutti i tweet passati e dalla rilevanza media dei tweet pubblicati dall'utente nell'ultima iterazione. Nell'esempio sopra è ben visibile, alla seconda riga, che il tweet pur non essendo rilevante è stato pubblicato da una fonte considerevole (si veda il valore 1 alla fine).

2. **Il calcolo dei pattern(getClueRelevance(tweet, interest)** : Per tutti i termini presenti nel tweet si crea una lista di pattern (ovvero un accostamento fra alcuni termini, degli n-grammi). Ad esempio:
carolina,democrats,realdonaldtrump=103,
america,carolina,north=103,
helping,house,walshfreedom=75,
clear,white,write=75,
Si fanno delle statistiche per capire se il pattern trovato è presente in altri tweet.
Il fine ultimo è di invidiare degli schemi ripetuti.
In base all'esempio citato prima, è facile che nei tweet recuperati si trovi l'accostamento dei termini "carolina,democrats,realdonaldtrump".

Aggiornamento delle statistiche

La classe **InterestUpdater** si occupa di aggiornare l'Interest con nuove frasi e hashtag in modo da massimizzare nelle prossime iterazioni l'acquisizione di nuovi tweet.
Alla fine della prima esecuzione della Java application per l'Interest "US Presidential Debate" la situazione è la seguente:



Sono stati recuperati 57 tweet di cui 50 rilevanti.
Alla fine della sessione l'InterestUpdater invoca la funzione update() che attraverso addNewPhrases(interest) genera delle nuove potenziali frasi. Ciò è possibile grazie ai pattern che sono stati giudicati rilevanti nel corso dell'acquisizione. Viene effettuato un calcolo sulla frequenza dei pattern, se questo calcolo supera il valore soglia 0.01 (fissato nel file config.txt) allora si crea una nuova frase con i termini presenti.
I dati recuperati sono poi soggetti ad una nuova analisi:
1. Viene calcolata la **frequenza** di ogni termine nelle frasi attraverso il metodo (getPhraseTermFreq)
2. Viene misurata la sua **specificità** attraverso il metodo getTermCommoness che utilizza un dump dei termini presenti in Wikipedia e ne individua la loro frequenza.Nel caso in cui il termine non sia presente nel dump, il valore di default è impostato a 0.5.
3. Viene assegnato un punteggio se questo termine può essere di rilievo per l'interest (termRelevanceToInterest) . Nell'immagine a seguire vi è un esempio di questi calcoli per il termine "final"

term	"final" (id=614)
interestTermFreq	0.0
termSpecificity	0.5
termRelevanceToInterest	0.25

Queste tre azioni vengono svolte per tutti i termini del pattern per ottenere un punteggio complessivo (score) che serve ad assegnare un peso. Se il peso supera la soglia dello 0.1 (questa soglia è fissata nel file config.txt con il nome di AcquisitionMinNewPhraseScore) allora viene aggiunto insieme al pattern alla lista delle potenziali frasi. In basso si può vedere l'esempio per il pattern costituito dai termini "commission debate"

key	Phrase (id=1324)
active	true
initial	false
lastUpdateTime	0
statistics	TotalStatistics (id=1333)
terms	HashSet<E> (id=1334)
text	"commission debate" (id=1325)
weight	0.25
value	Double (id=1331)

La stessa operazione viene poi fatta per gli hashtag. Le nuove potenziali frasi vengono ordinate a seconda dello score ottenuto e inserite nell'Interest, tuttavia non vengono persistite nel database.

```
[t_ac] INFO  qg.InterestUpdater - added phrase: commission debate score: 0.25
[t_ac] INFO  qg.InterestUpdater - added phrase: debate improvement score: 0.25
[t_ac] INFO  qg.InterestUpdater - added phrase: debate proudresister score: 0.25
[t_ac] INFO  qg.InterestUpdater - added phrase: debate realdonaldtrump score: 0.25
```

La sessione termina e viene stampato sulla console un report:

```
[t_ac] INFO  ta.Window - WTC: 57
[t_ac] INFO  ta.Window - WETC: 54
[t_ac] INFO  ta.Window - WRTC: 50
[t_ac] INFO  ta.Window - WAvgRel: 0.8662280701754387
[t_ac] INFO  ta.Window - WMinRel: -1.0
[t_ac] INFO  ta.Window - WRHC: 0
[t_ac] INFO  ta.Window - WIHC: 1
[t_ac] INFO  ta.Window - TUC: 54
[t_ac] INFO  ta.Window - TTC: 57
[t_ac] INFO  ta.Window - TRTC: 50
[t_ac] INFO  ta.Window - TDTC: 84
[t_ac] INFO  ta.Window - window has been shutdown.
```

Vengono indicati, in ordine, rispettivamente: il numero di tweet totali recuperati duranti la sessione, il numero di tweet in lingua inglese, il numero di tweet rilevanti nella sessione, il valore medio ed il valore minimo della

rilevanza, il numero di hashtag rilevanti durante la sessione, il numero di hashtag irrilevanti, il numero di utenti che sono stati analizzati, il numero totale di tweet, il numero totale di tweet rilevanti, il delta dei tweet.
Conclusa la sessione, automaticamente ne parte una nuova con una nuova lista di frasi

```
INFO  qg.QueryGenerator - Total Interest's Phrase Count: 51
INFO  qg.QueryGenerator - Number of phrases to exploit: 4
INFO  qg.QueryGenerator - Number of phrases to explore: 12
INFO  qg.QueryGenerator - Choosen Phrases are: [debate democrates, debate medium, agile debate, debate slurs, debate president
INFO  ta.Acquisition - Listening is started.
```

Nella seconda iterazione, le statistiche sono le seguenti:

```
ta.Window - window is full
ta.Window - window has been finished.
ta.Acquisition - window processing has been done.
ta.Window - WTC: 54
ta.Window - WETC: 52
ta.Window - WRTC: 50
ta.Window - WAvgRel: 0.8881859357209877
ta.Window - WMinRel: -1.0
ta.Window - WRHC: 6
ta.Window - WIHC: 0
ta.Window - TUC: 107
ta.Window - TTC: 110
ta.Window - TRTC: 100
ta.Window - TDTC: 31
ta.Window - window has been shutdown.
```

Si aggiorna nuovamente la lista di frasi e si apre una nuova sessione:

```
qg.QueryGenerator - Total Interest's Phrase Count: 51
qg.QueryGenerator - Number of phrases to exploit: 16
qg.QueryGenerator - Number of phrases to explore: 12
qg.QueryGenerator - Choosen Phrases are: [debate democrates, debate medium, agile debate, debate presidential,
```

I tempi di esecuzione sono molto lunghi. Nel caso di “US Presidential Debate” sono state impiegate circa due ore ma questo è dipeso dalla velocità del TwitterListener nell’acquisire i tweet.
Se ad esempio si sceglie un argomento che ha un maggiore risalto e che quindi ha continuamente nuovi tweet al riguardo (es.Donald Trump), l’esecuzione sarà molto più veloce.
Le frasi che ho utilizzato sono state: president USA e Donald Trump.
In basso gli screenshot dell’esecuzione con il calcolo della rilevanza per i nuovi pattern acquisiti:

```
rc.RelevanceCheck - donald trump ~       president,trump 0.12626262626262627
rc.RelevanceCheck - president usa        ~       president       0.45454545454545453
rc.RelevanceCheck - donald trump ~       president       0.12626262626262627
rc.RelevanceCheck - donald trump ~       stand   0.020202020202020204
rc.RelevanceCheck - donald trump ~       president,stand,trump   0.015151515151515152
rc.RelevanceCheck - donald trump ~       trump   0.5707070707070707
rc.RelevanceCheck - donald trump ~       join,trump      0.010101010101010102
rc.RelevanceCheck - donald trump ~       stand,trump     0.020202020202020204
rc.RelevanceCheck - donald trump ~       join    0.010101010101010102
rc.RelevanceCheck - donald trump ~       president,stand 0.015151515151515152
[t_ac] INFO  ta.Window - WTC: 236
[t_ac] INFO  ta.Window - WETC: 236
[t_ac] INFO  ta.Window - WRTC: 224
[t_ac] INFO  ta.Window - WAvgRel: 0.9094469792314726
[t_ac] INFO  ta.Window - WMinRel: -1.0
[t_ac] INFO  ta.Window - WRHC: 1
[t_ac] INFO  ta.Window - WIHC: 3
[t_ac] INFO  ta.Window - TUC: 866
[t_ac] INFO  ta.Window - TTC: 1066
[t_ac] INFO  ta.Window - TRTC: 889
[t_ac] INFO  ta.Window - TDTC: 998
[t_ac] INFO  ta.Window - window has been shutdown.
[t_ac] INFO  qg.QueryGenerator - Total Interest's Phrase Count: 33
[t_ac] INFO  qg.QueryGenerator - Number of phrases to exploit: 8
[t_ac] INFO  qg.QueryGenerator - Number of phrases to explore: 10
[t_ac] INFO  qg.QueryGenerator - Choosen Phrases are: [county shell tuesday, trump, donald, reminder soul, leaves trample, choice
```

Conclusioni

In questo documento, è stato trattata la raccolta adattiva di tweet per un argomento di interesse attraverso l'utilizzo delle API di Twitter. Ciò su cui il lavoro si basa è la selezione delle frasi che vengono elaborate tramite l'utilizzo di stopword senza stemmer. L'algoritmo avrebbe potuto generare risultati migliori attraverso il package topK e la libreria dimple: la loro implementazione infatti si basa sull'utilizzo di n-grammi.
Gli N-grammi frequenti hanno maggiori probabilità di essere frasi significative e seguono la distribuzione di zipf. In assenza di ciò l'algoritmo risulta meno preciso. La strategia utilizzata è una strategia di apprendimento con lo scopo di fornire una buona copertura coerente di tweet per un determinato argomento:
Viene mantenuta una stima del grado di importanza di ogni frase e dinamicamente viene accertata la pertinenza dei tweet in arrivo.
L'obiettivo è massimizzare il richiamo modificando automaticamente la query e aggiornando l'insieme di frasi che rappresentano l'algoritmo di interesse.
Man mano che le iterazioni vanno avanti otteniamo sempre più statistiche sulle frasi che sono state selezionate. Il grado di importanza di una frase cambia spesso e perciò serve un meccanismo adattivo che è proprio quello utilizzato in questo sistema di acquisizione.
Risultati simili possono essere ottenuti tramite le librerie Python,:Tweepy, Textblob (per l'analisi testuale) e NLTK(Natural Language Toolkit).
Con Textblob è possibile effettuare operazioni di tokenizzazione rimuovendo le parole comunemente usate e che sono irrilevanti nell'analisi del testo. E' possibile eseguire il tagging POS dei token e selezionare solo le porzioni di testo (aggettivi, avverbi, ecc.) che sono significativi.
Viene in questo modo resa più semplice l'elaborazione del testo e un' operazione simile può essere effettuata nel progetto corrente grazie all'uso di Stanford Core NLP.
CoreNLP comprende infatti part-of-speech (POS) tagging, entity recognition, pattern learning, parsing e permetterebbe al lavoro svolto da M.Sadri di riconoscere nomi di persona e luoghi .
Ciò consentirebbe di migliorare ulteriormente la ricerca e l'acquisizione dei tweet.