UNIVERSITÀ
DEGLI STUDI
DI PALERMO

Progetto e sviluppo di un sistema per l'analisi dei dati provenienti da social network

Tesi di Laurea Magistrale in Ingegneria Informatica

E. Simonelli
Relatore: Prof. Giuseppe Lo Re
Correlatore: Ing. Marco Morana

Progetto e sviluppo di un sistema per l'analisi dei dati provenienti da social network

Tesi di Laurea di
Dott. Eliseo Simonelli

Relatore
Ch.mo Prof Giuseppe Lo Re

Controrelatore
Ch.mo Prof. Salvatore Gaglio

Correlatore
Ing. Marco Morana

Sommario

La constante diffusione dei social network, specialmente con Twitter e Facebook, ha condotto ad un'era dove la produzione e la condivisione delle informazioni fa ormai parte della quotidianità. In particolare, i più importanti social network hanno permesso agli utenti di condividere enormi quantità di informazioni in cui esprimono i loro pareri riguardo ad eventi, anche in tempo reale, che accadono in tutto il mondo.

Per tale motivo, l'analisi dei dati su Twitter diventa interessante per poter estrarre delle informazioni utili, al fine di poterle utilizzare per la realizzazione di un sistema automatico volto non solo a catturare eventi in real time o come la gente reagisce in base a questi eventi, ma anche per poter informare l'utente finale su uno specifico evento. Di fatto, poter estrarre informazioni utili da questa grande mole di dati, contenenti un alto livello di rumore, diventa un processo molto oneroso e risulta essere vantaggioso applicare dei filtri al fine di eliminare delle informazioni meno importanti.

L'obiettivo principale della tesi consiste nell'analizzare lo stream di Twitter per rilevare argomenti rilevanti all'interno di un evento generico. Il sistema è stato creato per adattarsi al comportamento della specifica natura dei dati in entrata. In particolare, si è tentato di ridurre il più possibile questa mole di dati con rumore al fine di produrre un sistema completamente automatizzato che possa avvisare l'utente su un evento specifico riuscendo a scartare le informazioni meno rilevanti da quelle più rilevanti dal flusso continuo di dati.

Indice

1. Introduzione

La constante diffusione dei social network, specialmente con Twitter e Facebook, ha condotto ad un'era dove la produzione e la condivisione delle informazioni fa ormai parte della quotidianità. In particolare, i più importanti social network hanno permesso agli utenti di condividere enormi quantità di informazioni in cui esprimono i loro pareri riguardo ad eventi, anche in tempo reale, che accadono in tutto il mondo.

Tra i social network di maggior successo c'è sicuramente Twitter ed anzi, per molti aspetti, è quello che sta ottenendo più rilevanza all'interno della società. Twitter non è da considerarsi uno strumento totalmente personale. Infatti, i messaggi vengono condivisi pubblicamente e tutti gli utenti sono in grado di visualizzarli senza il consenso esplicito di chi ha prodotto l'informazione stessa. Il ruolo di Twitter e dei tweet è sempre più importante specialmente se usati per dichiarazioni di pubblico interesse, principalmente in caso di grossi avvenimenti.

Per tale motivo, l'analisi dei dati su Twitter diventa interessante per poter estrarre delle informazioni utili, al fine di poterle utilizzare per la realizzazione di un sistema automatico volto non solo a catturare eventi in real time o come la gente reagisce in base a questi eventi, ma anche per poter informare l'utente finale su uno specifico evento. Di fatto, poter estrarre informazioni utili da questa grande mole di dati, contenenti un alto livello di rumore, diventa un processo molto oneroso e risulta essere vantaggioso applicare dei filtri al fine di eliminare delle informazioni meno importanti.

L'obiettivo principale della tesi consiste nell'analizzare lo stream di Twitter per rilevare argomenti rilevanti all'interno di un evento generico. Il sistema è stato creato per adattarsi al comportamento della specifica natura dei dati in entrata. In particolare, si è tentato di ridurre il più possibile questa mole di dati con rumore al fine di produrre un sistema completamente automatizzato che possa avvisare l'utente su un evento specifico riuscendo a scartare le informazioni meno rilevanti da quelle più rilevanti dal flusso continuo di dati.

Questo lavoro di tesi è organizzato come segue:

- Una descrizione dello stato dell'arte del sistema preso in esame e degli algoritmi presenti in letteratura;

- Presentazione e descrizione degli algoritmi di Topic Detection;
- Analisi del sistema presentato nel punto precedente e rielaborazione per lo sviluppo del sistema web di ricerca e di Detection;
- Approfondimento e individuazione dei migliori criteri per la realizzazione del sistema di Acquisizione e di Detection;
- Elaborazione di un sistema di Live Detection in grado di poter raggruppare contemporaneamente la fase di Acquisizione e di Detection;
- Sviluppo di nuovi sistemi di analisi al fine di migliorare il risultato ottenuto dal punto precedente;
- Realizzazione di un sistema di valutazione automatizzato in grado di evidenziare le problematiche presentate nel corso dello sviluppo della tesi;
- Analisi dei risultati e confronto tra i vari metodi realizzati al fine di determinare quale di essi risulta essere il migliore.

2. Stato dell'arte

In questo capitolo si affronterà lo stato dell'arte relativamente alle principali tematiche affrontate nel corso della tesi.

Per mantenere il nostro approccio generale, consideriamo che lo stream in ingresso è fatto da piccoli pezzi di testo generati dagli utenti (nel caso di Twitter è composto da tweet). I *post* sono formati da una sequenza di parole, termini o keywords, e ognuno di essi è composto dalla sua data di creazione.

Ci concentriamo in uno scenario in cui l'utente avvia il sistema di *Detection* fornendo un set di chiavi di ricerca, che sono usate come filtro iniziale per limitare la ricerca sui post che contengono soltanto quelle determinate chiavi di ricerca.

Inoltre, anche se in alcuni sistemi presenti in letteratura [10] viene fornito l'intervallo di tempo di interesse e la frequenza di aggiornamento dei post (esempio: ogni 10 minuti), nel nostro sistema viene utilizzata una finestra temporale dinamica, la quale aggiorna automaticamente il periodo di tempo con cui vengono scaricati i tweet. Infatti, in [17] e [18], è stato dimostrato come la finestra temporale dinamica si adatta meglio alla ricerca di argomenti di interesse in tempo reale.

Come output dell'algoritmo avremo un serie di Topic, ciascuno composto da una lista di keyword, emessi alla fine di ogni intervallo di tempo e determinato dalla finestra temporale dinamica.

Questa impostazione si adegua molto bene allo scenario del mondo reale, ed anche se richiede un input iniziale da parte dell'utente, questo tipo di approccio rimane sempre generico e adatto ad ogni tipo di argomento o evento.

Una volta acquisito lo stream dei tweet, si passa ad analizzarlo tramite la Topic Detection. Per poter affrontare questo tipo di problema vi sono diverse tecniche, che possono essere approssimativamente suddivise in 3 categorie principali [1]:

1. *Document-pivot*: questi approcci creano gruppi di documenti secondo una specifica rappresentazione. Per esempio, ogni termine all'interno del documento può essere rappresentato dal suo valore di TF-IDF (*Term Frequency-Inverse Document Frequency*) [2].

1. *Probabilistic Topic Model*: questi approcci partono dal presupposto che esistono sempre alcuni Topic latenti, quindi si riferiscono ad algoritmi statistici per scoprire le strutture semantiche latenti di un vasto corpo di testo. L'algoritmo più utilizzato appartenente a questa categoria è l'algoritmo LDA (*Latent Dirichlet Allocation*) [6].
2. *Feature-pivot*: questi approcci creano cluster di termini calcolando i modelli di co-occorrenza tra coppie di termini selezionati fra diversi documenti. Questi metodi differiscono principalmente dagli altri per il loro meccanismo di selezione dei termini che usano.

Gli approcci generali discussi finora, spesso rappresentano le basi di diversi lavori che analizzano i dati estratti dai social network per individuare argomenti di tendenza. Questi metodi presentano sia dei vantaggi che degli svantaggi e nel paragrafo successivo verranno approfonditi questi aspetti.

2.1. Data Pre-Processing

Il contenuto dei messaggi generati dagli utenti poterebbe presentare un elevato numero di rumore. Per ridurre questa quantità di rumore prima che venga eseguita la *Detection*, i dati raccolti attraverso la fase di ricerca potrebbero essere sottoposti ad una fase di pre-processing per eliminare informazioni superflue. Questi step sono:

- *Tokenizzation*: si utilizza questa fase per poter estrarre gruppi di termini scritti in forma corretta dal messaggio originale eliminando le cosiddette *stop words,* eliminando la punteggiatura, riducendo i caratteri ripetuti e rimuovere le menzioni.
- *Stemming*: è il processo mediante il quale si riducono le parole alla loro forma radice (o tema), cosi da poter mettere in relazione le parole che presentano la stessa radice. Questo processo tenta di riconoscere la parte flessa della parola tentando di eliminarla, ma risulta essere fortemente legata al tipo di linguaggio preso in esame, molti linguaggi diversi dall'inglese presentano difficoltà maggiori nel riconoscere questi termini, per questo in letteratura si fa riferimento sempre alla lingua inglese.

- *Aggregation*: è il processo mediante il quale si aggregano più messaggi tra di loro al fine di creare dei *super documenti* di grandi dimensioni. Questi *super documenti* vengono costruiti o aggregandoli in base all'allocazione temporale, che attacca *n* messaggi tra loro se risultano essere contigui nel tempo, oppure aggrega tutti i messaggi che sono duplicati all'interno dello stesso intervallo di tempo, identificati tramite un metodo molto efficiente presentato in [13]. Questo processo, in alcuni casi, risulta essere inconveniente in quanto introduce il rischio di accorpare termini correlati che appartengono ad argomenti differenti.

Questa fase di *data pre-processing*, come descritto in [17] e [18], non porta alcun vantaggio in termini di prestazioni, anzi risulta peggiorarle, motivo per il quale in questo sistema non vengono utilizzate.

2.2. Document-Pivot

Gli algoritmi che appartengono a questa classe creano gruppi di documenti sfruttando una certa metrica di somiglianza. Il lavoro di Phuvipadawat e Murata [7] segue questa direzione per fornire un metodo per rilevare *breaking news* da Twitter. Una volta recuperati i tweet utilizzando query mirate, essi vengono convertiti in gruppi di parole in base al loro valore di *TF-IDF* (T*erm Frequency - Inverse Document Frequency*), sottolineando entità importanti come un nome di un paese o personaggi pubblici. I tweet sono poi raggruppati incrementalmente in base alla loro similarità testuale tra quelli in arrivo e quelli già esistenti.

Approcci simili basati sulla similarità testuale e dal loro valore *TF-IDF* possono essere trovati in letteratura [8], [9]. Tra loro, il metodo discusso da Becker in [9], considera inoltre la classificazione dei tweet in base al fatto che essi appartengono ad eventi del mondo reale o no. Un importante inconveniente di questo metodo è la necessità di un'annotazione manuale dei campioni di training e di test.

Una dimensione del testo diversa può essere usata anche per migliorare la qualità del raggruppamento. TwitterStand [10] usa un algoritmo di raggruppamento chiamato "*leader-follower*", che prende in considerazione sia la loro similarità testuale che la loro vicinanza temporale. Ogni gruppo è rappresentato dal vettore del baricentro della *TF-IDF* e dal tempo medio del post. Gli inconvenienti di questo

metodo sono la sensibilità al rumore e alla frammentazione dei gruppi di parole, che risultano essere un tipico problema per i metodi *document-pivot* [11].

Il lavoro fatto da Petrovic al. [13] è strettamente collegato al metodo *document-pivot*, l'obiettivo è quello di individuare una discussione interessante in una grande insieme di tweet. In questo elaborato viene utilizzato l'algoritmo *Local Sensitive Hashing* (LSH), tuttavia questa soluzione risulta essere problematica quando le parole di ricerca non sono molto simili.

La grossa mole di dati da elaborare, principalmente il calcolo della distanza fra gli oggetti (*item*) di un insieme di dati, è un grosso vincolo allo sviluppo di applicazioni in tempo reale per soddisfare la similarità fra parti di testo o di documenti. Quindi, questo algoritmo, risulta essere un metodo efficiente per ridurre la dimensione degli spazi vettoriali in modo da rendere più immediata la ricerca di parole simili.

Il *Local Sensitive Hashing* si basa nell'applicazione di una funzione di *hash* ai *termini* in input in modo da far collidere, con alta probabilità, *termini* simili negli stessi contenitori (*bucket*). Quando arriva un'altro *termine*, dapprima viene inserito nel contenitore di appartenenza, poi viene esaminato e l'algoritmo restituisce il più vicino. Funziona come segue:

- Confronta la metrica di somiglianza della rappresentazione *TF-IDF* del post in entrata, con la metrica di tutti gli altri post già acquisiti.
- Se la metrica, del miglior post individuato, supera la soglia θ_{TF-IDF}, assegna il *termine* allo stesso gruppo di post e lo identifica come il migliore; altrimenti crea un'altro gruppo dove il nuovo post trovato è l'unico elemento.

Poiché si tratta di documenti testuali, una misura particolarmente interessante di distanza è il coseno dell'angolo tra le rappresentazioni vettoriali dei due documenti. Allan in [12] riporta come questa metrica sia migliore rispetto alla divergenza di *Kullbach-Leiblar* , alla S*omma Pesata* o alla *Language Model*.

- *Coseno*: la misura è semplicemente il prodotto interno di due vettori, in cui ciascun vettore è normalizzato per unità di lunghezza. Essa rappresenta il coseno dell'angolo tra due vettori *d* e *q*.

$$\frac{\left(\sum q_i d_i\right)}{\sqrt{\left(\sum q_i^2\right)\left(\sum d_i^2\right)}}$$

Dove q e d hanno lunghezza unitaria. La similitudine del coseno tende a funzionare meglio quando si raggiunge la massima dimensionalità. Le performance degradano se uno dei due vettori diventa più piccolo.

- *Divergenza di Kullbach-Leiblar*: (anche detta *divergenza di informazione*, e*ntropia relativa*, o *KLIC*) è una misura non simmetrica della differenza tra due distribuzioni di probabilità P e Q. Specificamente, è la misura dell'informazione persa quando Q è usata per approssimare P. Anche se è spesso pensata come una distanza, non è una vera e propria metrica, infatti, non è simmetrica: la KL da P a Q non è in genere la stessa KL da Q a P:

$$D_{KL}(P \parallel Q) = \sum_i \ln\left(\frac{P(i)}{Q(i)}\right) P(i)$$

- *Somma Pesata*: rappresenta una combinazione lineare di evidenza con pesi e rappresenta l'attendibilità associata con diversi elementi di evidenza.

$$\frac{\left(\sum q_i d_i\right)}{\left(\sum q_i\right)}$$

dove q_i rappresenta il vettore della *query*, mentre d_i rappresenta il vettore del documento.

La somma pesata tende a rendere meglio quando il vettore della query q_i ha una bassa dimensionalità. Infatti, è stata escogitata principalmente per dare un vantaggio per piccole richieste da parte degli utenti.

- *Language Model*: è una distribuzione di probabilità su una sequenza di parole. Esso fornisce un approccio probabilistico per calcolare la similarità tra un documento e un Topic o tra due documenti. Rappresenta modelli di parole usate e stima quale modello (se esiste) è il migliore che ha generato rispetto a quelli nuovi in arrivo. Precisamente:

$$\frac{P(M \mid D)}{P(D)} = \frac{\prod_i P(d_i \mid M)}{\prod_i P(d_i)}$$

dove d_i rappresenta il singolo termine all'interno del documento D, ed M è preso come modello da uno dei documenti già analizzati. Così si usa uno stimatore di massima verosimiglianza, il quale è semplicemente il numero delle occorrenze di d_i in M diviso per il numero totale dei termini in M.

Questo è il motivo per cui Charikar in [14] utilizza il coseno dell'angolo tra le due rappresentazioni vettoriali, in cui la probabilità della collisione tra due punti è proporzionale al coseno dell'angolo tra di essi. Questo schema è stato utilizzato per creare una lista di somiglianza di sostantivi da Ravichandran in [15]. Più precisamente, la probabilità di collisione tra di loro di due punti x e y è:

$$P_{coll} = 1 - \frac{\theta(x,y)}{\pi}$$

Dove $\theta(x,y)$ è l'angolo tra x e y. Se si utilizza più di un iperpiano, si è in grado di diminuire la probabilità di collisione con i punti non simili. Il numero degli iperpiani k può essere considerato come il numero di bit per ogni termine nel loro schema di *hash*. In particolare:

$$\text{se } x \bullet u_i < 0, i \in [1...k]$$

per un documento x e un vettore dell'iperpiano u_i, noi settiamo l'*i-mo* bit a 0, altrimenti 1. Dato il numero desiderato di bit *k*, e la probabilità desiderata di perdita dei vicini δ, si può calcolare il numero delle tabelle di *hash* L come:

$$L = \log_{1-p_{coll}^{k}} \delta$$

2.3. Probabilistic Topic Model

L'estrazione di argomenti interessanti in numerosi documenti di testo può essere affrontato attraverso la *Probabilistic Topic Model*. In generale, un *modello di Topic*, è un modello Bayesiano [29,30] che associa ad ogni documento una certa distribuzione di probabilità in base agli argomenti, che a sua volta sono su una distribuzione dei termini più frequenti. Il *Latent Dirchlet Allocation* (*LDA*) [6] è uno tra gli algoritmi più conosciuti e utilizzati di questa categoria.

Secondo *LDA*, ciascun documento è considerato un gruppo di termini, i quali, sono le uniche variabili osservate nel *modello*. La distribuzione dei *Topic* per ciascun documento e la distribuzione dei *termini* per ciascun *Topic* vengono nascoste e devono essere stimate tramite l'inferenza Bayesiana.

LDA assume il seguente processo di generazione per ogni documento *w* e in un corpo *D*:

1. Prendi $N \sim Poisson(\xi)$
2. Prendi $\theta \sim Dir(\alpha)$
3. Per ogni N termini w_n
    3.1. Prendi un topic $z_n \sim Multinomial(\theta)$
    3.2. Prendi un termine W_n da $p(w_n \mid z_n, \beta)$, probabilità condizionata sul topic z_n

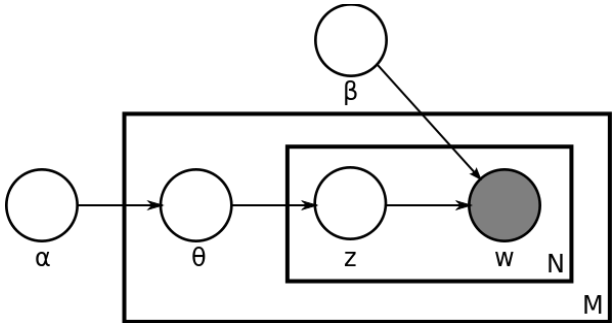


Fig. 1 Rappresentazione grafica del modello utilizzato in LDA.

Da questo modello di base vi sono diverse ipotesi per poterlo semplificare:

1. La dimensionalità k della distribuzione di Dirichlet (e quindi la dimensionalità del topic z) si assumono fisse e conosciute.
2. La probabilità del termine è parametrizzata da una matrice $\beta = K \times V$ dove $\beta_{ij} = p(w^j = 1 \mid z^i = 1)$, che per ora trattiamo come una quantità fissa che deve essere stimata.
3. La distribuzione di *Poisson*, è una distribuzione di probabilità discreta che esprime le probabilità per il numero di eventi che si verificano successivamente ed indipendentemente in un dato intervallo di tempo, sapendo che mediamente se ne verifica un numero λ. Per tale motivo, risulta non essere critica per questo tipo di calcolo.
4. Si può notare come N sia indipendente rispetto a tutte le altre variabili generate (θ ed anche z), questo significa che è una variabile accessoria e generalmente si può ignorare.

Una variabile casuale θ, *k-dimensionale* di Dirichlet, se $\theta \geq 0$, allora $\sum_{i=1}^{k} \theta_i = 1$, e ha la seguente densità di probabilità:

$$p(\theta \mid \alpha) = \frac{\Gamma(\sum_{i=1}^{k} \alpha_i)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \theta_i^{\alpha_i - 1} ... \theta_k^{\alpha_k - 1} \ \Gamma$$

dove il parametro α è un k-vettore con componenti $\alpha_i > 0$, e dove $\Gamma(x)$ è la funzione *Gamma*.

Dati i parametri α e β, data la probabilità congiunta di un Topic composto θ e un gruppo di N Topic z, il gruppo di N termini w estratti è dato da:

$$p(\theta, z, w \mid \alpha, \beta) = p(\theta \mid \alpha) \prod_{n=1}^{N} p(z_n \mid \theta) p(w_n \mid z_n, \beta)$$

dove $p(z_n \mid \theta)$ è semplicemente θ_i tale che $z_n^i = 1$. Integrando su θ e sommando su z, si ottiene la distribuzione marginale di un documento:

$$p(w \mid \alpha, \beta) = \int p(\theta \mid \alpha) \left(\prod_{n=1}^{N} \sum_{z_n} p(z_n \mid \theta) p(w_n \mid z_n, \beta) \right) d\theta$$

Infine, prendendo il prodotto delle probabilità marginali di ogni singolo documento, si ottiene la probabilità di un *corpo:*

$$p(D \mid \alpha, \beta) = \prod_{d=1}^{M} \int p(\theta_d \mid \alpha) \left(\prod_{n=1}^{N_d} \sum_{z_{dn}} p(z_{dn} \mid \theta_d) p(w_{dn} \mid z_{dn}, \beta) \right) d\theta$$

L'algoritmo *LDA* è rappresentato da un modello come in Figura 2. Come rende chiaro la figura, ci sono tre livelli che rappresentano *LDA*. I parametri α e β sono i parametri del *corpus-level,* presi durante il processo di generazione del corpus. La variabili θ_d sono variabili del document-level, prese una per ogni documento. Infine, le variabili z_{dn} e w_{dn} sono variabili del word-level e sono prese una volta per ogni parola presente in ogni documento.

Questo algoritmo risulta essere molto efficiente, in quanto, non consuma molte risorse riuscendo ad avere ottime prestazioni.

2.4. Feature-Pivot

I metodi cha appartengono alla categoria *Feature-Pivot* raggruppano insieme i termini in base al loro modello di *co-occorenza* all'interno dei documenti. Per esempio, la tecnica proposta in [3] usa FP-Growth [4] in parallelo, per selezionare set di parole frequenti. Poiché in diversi casi considerare le co-occorrenze tra coppie

di termini può essere una limitazione, approcci come l'algoritmo FPM (*Frequent Pattern Mining*) [5] sono basati nell'analisi delle co-occorrenze tra qualsiasi numero di termini.

2.4.1. FP-Growth

Questo algoritmo, proposto da Han in [14], è un metodo efficiente e scalabile per l'estrazione di un set completo di termini frequenti all'interno di un documento, utilizzando una struttura a grafo, per la memorizzazione di informazioni complesse e cruciali riguardo *pattern* frequenti, chiamata "*Frequent-Pattern Tree*" (*FP-tree*).

La struttura *FP-Tree,* è una struttura compatta che memorizza una moltitudine di informazioni riguardo alla frequenza dei modelli all'interno di un database. In particolare:

1. Una radice etichettata "*null*" con un insieme di nodi figli, e una tabella degli elementi frequenti in cui si riporta ogni nodo.

2. Ogni nodo è composto da tre campi: *item-name*, *count*, e *node-link;* dove *item-name* identifica quale elemento rappresenta il nodo, *count* identifica il numero di operazioni per raggiungere il nodo (il costo), *node-link i*ndica il collegamento al nodo successivo, se esiste altrimenti è posto a *null*.

3. Ogni riga della tabella è composta da due campi: *item-name*, lo stesso del nodo; *head of node-links,* che punta al primo nodo successivo nel grafo. In aggiunta la tabella può anche riportare il campo *count* di ogni elemento (vedi Figura 1).

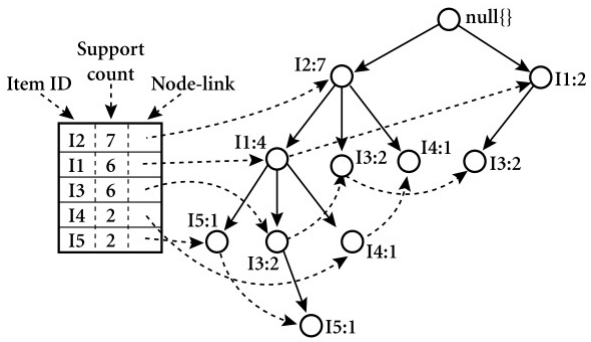


Fig. 2 Esempio di un FP-Tree.

Questo tipo di struttura viene costruita tramite due ricerche nel database. La prima ricerca raccoglie e ordina i set dei termini frequenti, mentre la seconda costruisce la struttura *FP-Tree*.

Dopo la costruzione di questa struttura è possibile trovare dei set completi di termini frequenti attraverso l'uso di *FP-Growth*. Questo algoritmo cerca ricorsivamente sulla struttura *FP-Tree*, estraendo ricorsivamente termini dall'albero di ricerca. Quindi, l'algoritmo *FP-Growth* si può riassumere in due passaggi principali:

1. Costruisce una struttura dati complessa chiamata *FP-Tree*;
2. Estrae i termini frequenti direttamente dalla struttura FP-Tree appena creata;

2.4.2. Graph-Based Feature-Pivot

Questo è il primo di una serie di algoritmi appartenenti alla classe *Feature-Pivot*. La sua caratteristica unica è che, per il raggruppamento dei termini, utilizza lo *Structural Clustering Algorithm for Networks* (*SCAN*) [15]. La proprietà che contraddistingue lo *SCAN* da altri tipi di algoritmi è che, oltre a rilevare una *comunità* di nodi che si accomunano per la presenza di termini frequenti, fornisce un elenco di *hub*, i quali ognuno di loro possono essere connessi ad una serie di questi nodi. Negli approcci di tipo *Feature-Pivot* per la *Topic Detection*, i nodi del grafo possono corrispondere ai termini e le *comunità* possono corrispondere agli argomenti da estrarre. Gli *hub* trovati posso essere idealmente considerati come dei termini che sono presenti in più argomenti di interesse.

Questo processo, solitamente, non è facile da raggiungere con un semplice algoritmo di raggruppamento, ma si deve fornire esplicitamente una connessione efficace tra gli argomenti. Di fatto, questo, rappresenta il motivo principale per cui viene preferito lo *SCAN*.

Cosi, si selezionano i termini da raggruppare, dalla serie di termini presenti nel corpo del documento, usando l'approccio discusso in [8]. Esso usa un corpo di riferimento indipendente, composto da una collezione casuale di tweet. Per ogni termine presente nel corpo di riferimento la probabilità di comparsa è calcolata cosi:

$$p(w \mid corpus) = \frac{N_w + \delta}{\left(\sum_u^n N_u\right) + \delta n}$$

dove N_w è il numero delle volte in cui il termine w compare nel corpo di riferimento, n è il numero dei tipi di termini che sono presenti nel corpo e δ è una costante (tipicamente 0.5) che viene inclusa per regolarizzare la stima di probabilità. Per determinare i termini più importanti nel nuovo corpo, si calcola il rapporto della probabilità di comparsa di ogni termine in tutti e due i corpi:

$$\frac{p(w \mid corpus_{new})}{p(w \mid corpus_{ref})}$$

I termini con il valore più alto saranno coloro che appariranno più frequentemente e ci si aspetta che essi siano correlati agli argomenti di maggior interesse nel documento. Nel dettaglio l'algoritmo funziona come segue:

1. *Selezione*: vengono selezionati tutti i k termini usando il rapporto di probabilità precedentemente descritto e per ogni termine si crea un nuovo nodo e gli archi collegano i due termini considerati simili;
2. *Collegamento*: inizialmente, viene selezionato una valore di similarità per ogni coppia di termini e solo successivamente viene calcolato questo valore;
3. *Raggruppamento*: l'algoritmo *SCAN* viene applicato al grafo, quindi viene generato un nuovo *Topic* per ogni argomento di interesse trovato;

Chiaramente, il processo di collegamento risulta essere cruciale per il corretto funzionamento del metodo. Un tipico problema con questo tipo di metodo è che al fine di raggruppare i termini comuni, si prendono in considerazione solamente le loro somiglianze a coppia, che si basano su diverse funzioni che calcolano il numero di occorrenze delle coppie di termini.

2.4.3. Frequent Items Mining

Nel caso in cui si trovano argomenti strettamente interconnessi tra loro che condividono un numero relativamente grande di termini simili, questa procedura risulta essere in grado di produrre argomenti molto più generici e quindi di accorpare un numero maggiore di *Topic* che trattano argomenti simili. Questa è la motivazione per la quale ci ha portato a considerare l'uso del *Frequent Itemset Mining* (*FIM*), ottima tecnica, inizialmente sviluppata per l'analisi di mercato, al fine di determinare quali elementi sono probabilmente più presenti in un gruppo di *transazioni* [16].

Nel contesto dei social network, un elemento è qualsiasi termine *w* menzionato in un post (escludendo punteggiatura, stop word). La *transazione* è il *post* e un gruppo di *transazioni* sono tutti i *post* che si presentano in un lasso di tempo T_j. Il numero di volte in cui un dato insieme di termini si ripresentano in un certo lasso di tempo è definito *supporto* ed ogni gruppo di elementi che soddisfa il supporto è chiamato *pattern.*

Quindi per ogni intervallo di tempo viene applicato un'algoritmo di *Frequent Pattern Detection* su un vasto numero di post e successivamente si utilizza il *FIM* per trovare gruppi di termini che più rappresentano argomenti di interesse per ogni intervallo di tempo selezionato. Questo set di parole estratte può essere considerato, tra tutti, quello che maggiormente rappresenta gli argomenti di interesse estratti.

2.4.4. Soft Frequent Pattern Mining

Il *Soft Frequent Pattern Mining* (*SFMP*) è la versione "Soft" del *FIM* , che considera sempre un numero di co-occorrenze dei termini maggiore di due, ma differentemente non richiede che tutti i termini co-occorrano frequentemente.

L'approccio proposto funziona mantenendo un gruppo di termini *S,* in cui i nuovi termini vengono aggiunti in base a come spesso si ripresentano in *S*. Per quantificare il numero di co-occorrenze reali tra un gruppo *S* di termini e un termine *t*, selezionato come candidato, si conserva un vettore D_s per *S* e un vettore D_t per il termine *t*, entrambi con lunghezza *n*, dove *n* è il numero di documenti nella collezione. L'*i-mo* elemento di D_s denota quanti dei termini in *S* co-occorrono nell'*i-mo* documento, mentre l'i-mo elemento di D_t è un'indicatore binario che rappresenta se i termini co-occorrono nell'*i-mo* documento oppure no. Così, il vettore D_t per un termine *t* che co-occorre frequentemente con i termini nel gruppo *S* avrà un valore elevato di similarità con il vettore corrispondente D_s.

Si può notare come alcuni termini presenti in D_s possono avere il valore di $|S|$, ciò significa che tutti i termini in *S* co-occorrono nei documenti corrispondenti, mentre altri possono avere un valore più piccolo che indica che soltanto un sottogruppo dei termini in *S* co-occorrono nei documenti corrispondenti. Per il termine che viene selezionato per espandere *S* è chiaro che ci saranno diversi contributi, sia da parte dei documenti in cui non co-occorrono tutti i termini, sia in quelli in cui co-occorrono tutti i termini.

In questo modo si ottiene la versione “Soft”, cioè nella modalità con cui viene scelto un termine per l’espansione e su come viene scelto un gruppo *S*. Infine, bisogna scegliere un criterio adeguato per arrestare l’algoritmo, il che significa, che se non viene correttamente impostato gli argomenti risultanti possono essere o troppo generici (poche keywords) oppure essere un mix di argomenti non interessanti (molte keywords collegate ad argomenti non rilevanti).

Per far fronte a questo, si usa come metrica il coseno per valutare la somiglianza tra *S* e il nuovo termine corrispondente. Se il valore è sopra una certa soglia il termine viene aggiunto, altrimenti il processo di espansione viene fermato. Questa soglia è l’unico parametro di questo algoritmo ed è dettato in modo da essere una funzione della cardinalità di *S*. In particolare si usa la funzione *Sigmoide* nella forma:

$$\theta(S) = 1 - \frac{1}{1 + e^{\frac{|S| - b}{c}}}$$

dove i parametri b e c possono essere usati per controllare la dimensione dei gruppi dei termini. In pratica, si setta il valore di questi due parametri cosicché la somma dei termini, quando la cardinalità di *S* è piccola, è semplice (la soglia è bassa), altrimenti la somma dei termini diventa difficile quando la cardinalità aumenta.

3. Twitter Live Detection Framework

In questo capitolo verrà mostrato brevemente l'algoritmo che è stato utilizzato come base su cui fondare lo studio progettuale e la realizzazione sperimentale del lavoro svolto.

Come precedente menzionato, il maggior vantaggio di adottare un approccio basato sull'algoritmo *SFPM* è dato dalla capacità di gestire un numero di co-occorrenze maggiore di due, senza richiedere che tutti i termini co-occorrano frequentemente. In questo elaborato, dato che gli argomenti rilevanti cambiano rapidamente, il sistema di ricerca in tempo reale deve riuscire ad adattarsi al cambiamento continuo dei dati e alla quantità dei dati in ingresso per fornire ottimi risultati.

Questo è il motivo principale il quale ci ha portato ad utilizzare l'algoritmo *TLDF* (*Twitter Live Detection Framework*) presentato in [17] e [18], una versione modificata del *Soft Frequent Pattern Mining* presentato nel Capitolo 1*,* che ora verrà spiegato in maniera dettagliata.

3.1. Vocabolario

Come primo passo il *Twitter Live Detection Framework* procede nella costruzione di un vocabolario, contenente tutti i termini che si presentano frequentemente nell'insieme dei tweet in esame. In [17] e [18] è stato dimostrato che la fase preliminare di *data pre-processing*, quindi l'eventuale uso dei metodi di *Tokenizzation* e *Stemming*, non portano alcuni benefici all'algoritmo, anzi ne peggiorano le prestazioni finali. Quindi, per lo sviluppo del sistema è stato scelto di non utilizzare questa fase di data pre-processing.

Il *TLDF* riceve in ingresso un insieme di tweet provenienti dalla prima fase di acquisizione, tramite l'avvio manuale della ricerca con le keyword da trovare. Successivamente, si passa alla costruzione di un vocabolario contenente tutti i termini frequenti individuati nel documento in esame.

Completato questo processo, il vocabolario viene riordinato portando in cima alla lista i termini ritenuti più importanti per poi andare ad individuare i relativi argomenti di interesse. Quindi, risulta importante capire come questi termini

vengano ritenuti di interesse per l'argomento in esame e come vengano selezionati. Di seguito se ne darà una piccola panoramica.

3.2. Metodo per la selezione dei termini

Il metodo di selezione dei termini adottato da *SFPM* permette di identificare i termini più importanti in un vocabolario di riferimento. Tuttavia, in uno scenario dinamico, l'attuale serie di termini per l'n-ma finestra W_n dovrebbe dipendere dai Topic trovati nella finestra W_{n-1}. Questa dipendenza causa che i termini già esistenti vengono preferiti rispetto ai termini emergenti che stanno crescendo all'interno della finestra corrente.

Al fine di evitare un tale comportamento, il valore di *likelihood ratio* usato da SFPM è stato combinato con la *TF-IDF* (*Term Frequency - Inverse Document Frequency*), una misura che sottolinea anche l'importanza di termini rilevanti nella finestra corrente. Infine, è stato aggiunto anche il *NER* (*Named-Entity Recognition*) per testare l'appartenenza dei termini su tre classi rilevanti chiamate: *persone*, *organizzazioni* e *luoghi.*

Cosi, il metodo di selezione dei termini utilizzato da Twitter Live Detection Framework prende i K termini con il più alto valore di *f:*

$$f(t) = w(t) \times r(t) \times TF - IDF(t)$$

dove:

- *w(t) = 1.5* se il termine *t* viene riconosciuto dal NER, altrimenti *w(t)=1*;
- *r(t)* è il valore di *Likelihood Ratio* del termine *t*;

Nella sezione successiva verrà spiegato brevemente come funzionano le tre componenti appena introdotte.

3.2.1. Likelihood Ratio

Dato un set attuale di tweet C_{cur}, il primo passo dell'algoritmo consiste nel selezionare i K termini più rilevanti t_k, che sono i termini con la più alta probabilità di presentarsi in C_{cur} e in un set di riferimento di tweet collezionati a caso C_{ref}:

$$r(t_k) = \frac{p(t_k \mid C_{cur})}{p(t_k \mid C_{ref})}$$

Questo è il valore che utilizza l'algoritmo SFPM per selezionare i termini che si ripresentano più spesso all'interno dell'insieme dei tweet in esame. Nel seguente lavoro, i tweet vengono acquisti in finestre temporali differenti, ma contigue. Per tale motivo, i tweet appartenenti ad un certo argomento non restano confinati all'interno della stessa finestra, specialmente se sono di piccola dimensione, ma tenderanno a ripresentarsi anche nelle finestre successive.

Tutto questo può portare a riconoscere stessi argomenti di interesse su più finestre temporali diverse e contemporaneamente ad omettere altri argomenti che sono più rilevanti per la finestra attuale. È stato dimostrato in [17] e [18] che la sola *likelihood ratio* non è in grado di discriminare correttamente tutti i termini, portando sempre alla composizione degli stessi argomenti.

Per rimuovere questo problema in [17] e [18] è stato scelto di pesare il valore di *likelihood ratio* con quello della funzione *TF-IDF*, presentato nel paragrafo successivo.

3.2.2. Term Frequency - Inverse Document Frequency

La funzione di peso *TF-IDF* viene utilizzata per misurare l'importanza di un termine rispetto ad un post (o tweet) o una collezione di post. Tale funzione aumenta proporzionalmente con il numero di volte che il termine è contenuto all'interno del post, ma cresce in maniera inversamente proporzionale con la frequenza del termine nella collezione. L'idea di base di questo comportamento è di dare più importanza a termini che compaiono nel documento, ma che in generale sono poco frequenti. Abbiamo quindi che:

$$(TF - IDF)_{i,j} = TF_{i,j} \times IDF_i$$

Il primo fattore della funzione è il numero dei termini presenti nel documento. In genere, questo numero viene diviso per la lunghezza del documento stesso per evitare che siano privilegiati documenti più lunghi.

$$TF_{i,j} = \frac{n_{i,j}}{|d_j|}$$

Dove $n_{i,j}$ è il numero di occorrenze del termine t_i nel documento d_j, mentre il denominatore è semplicemente la dimensione, espressa in numero di termini, del documento.

Il secondo fattore della funzione indica l'importanza generale del termine nella collezione:

$$IDF_j = \log \frac{|D|}{|\{d : t_i \in d\}|}$$

Dove $|D|$ è il numero di documenti nella collezione, mentre il denominatore è il numero di documenti che contengono il termine t_i.

Ponderando il rapporto di verosimiglianza (*Likelihood Ratio*) $r(t)$, con la $TF - IDF(t)$, permette di filtrare i termini comuni e selezionare i termini che sono rilevanti sia nella collezione (nella finestra precedente), sia nella finestra corrente.

3.2.3. Named Entity Recognition (NER)

La *Named Entity Recognition (NER)* è un'attività secondaria per l'estrazione di informazioni che cerca di individuare e classificare i termini in categorie predefinite come i nomi di persone, organizzazioni, luoghi, espressioni del tempo, quantità, valori monetari, percentuali, ecc.

Cosi la *Stanford University* [19], ha sviluppato un software in Java che implementa il *NER*. Questo software, rilasciato sotto forma di libreria per Java riconosce entità nominali cioè termini e li classifica assegnandogli un'etichetta, per esempio, se riconosce il nome di una persona gli assegna "person". Quindi esso come input riceve un termine e come risultato si ottiene una stringa riportante una delle possibili etichette.

Vi sono diverse configurazione di questo software, ma quella che meglio si adatta a questo tipo di lavoro, dimostrato in [17] e [18], risulta essere quella che produce come risultato:

- PERSON: il nome di una persona;
- LOCATION: il nome di un luogo;
- ORGANIZZATION: il nome di una organizzazione;
- 0, altrimenti;

Questa configurazione permette che i risultati siano inerenti alle classi generali utilizzate comportando, quindi, un risultato generico ma più preciso.

In questo elaborato, riconoscere alcuni termini come i nomi di persone, i quali hanno un significato maggiore rispetto ad altri termini, permette di dare un

peso maggiore a termini che fanno parte di argomenti di interesse rispetto a quelli che non lo sono.

L'unico inconveniente di questo metodo è che dipende dalla lingua dei messaggi trattati. Infatti, il NER attualmente supporta l'inglese, il tedesco, lo spagnolo e il cinese. Anche se questo fattore limita la ricerca dei termini nei linguaggi spora citati, in questo momento per la sperimentazione di questo sistema risulta pressoché adeguato, in quanto si è scelto di trattare soltanto messaggi in lingua inglese. Inoltre, l'importanza dei termini trovati dal NER viene poi amplificato con un fattore di 1.5, come spiegato in [7].

3.2.4. Meccanismo di selezione dei termini

Il metodo di selezione dei termini viene eseguito al primo passo dell'algoritmo *TLDF*, al fine di selezionare i termini che si ripetono frequentemente all'interno dell'insieme dei tweet in esame. Quindi, questo algoritmo calcola il valore di *likelihood ratio* dei termini nel vocabolario appena creato p_{new} e nel vocabolario di riferimento p_{ref} rispettivamente. Successivamente, calcola il valore $r(t)$ con cui questi termini si presentano. Infine, se il termine viene trovato dal NER assegna a $w(t)$ il valore di 1,5, altrimenti 1.

Si presenta di seguito lo pseudocodice dell'algoritmo utilizzato per la selezione dei termini:

Meccanismo di selezione dei termini TLDF_*TermSelection*(*C*,*K*):

1. Per ogni termine *t* in *C:*
    - 1.1. $p_{new} = LikelihoodOfAppearance(t, C_{new})$
    - 1.2. $p_{ref} = LikelihoodOfAppearance(t, C_{ref})$
    - 1.3. $r(t) = p_{new} / p_{ref}$
    - 1.4. $TF - IDF(t) = ComputeTFIDF(t)$
    - 1.5. Se il termine *t* è riconosciuto dal *NER:*
        - 1.5.1. $w(t) = 1.5$
    - 1.6. Altrimenti:
        - 1.6.1. $w(t) = 1$
    - 1.7. $f(t) = w(t) \times r(t) \times TF - IDF(t)$

2. Riordina *f(t)* in ordine crescente;
3. $T = 0$
4. Per *i* che va da 1 a *K*:
5. $T = T \cup t(f_i)$
6. Ritorna *T*

L'algoritmo ritorna la lista ordinata dei termini che si presentano più frequentemente all'interno dei tweet scaricati e scelti come argomenti rilevanti.

3.3. Scelta della finestra di ricerca

Un'ulteriore considerazione riguarda il modo in cui viene scelta la dimensione della finestra di ricerca nel *TLDF*. L'utilizzo di finestre di ricerca con dimensione fissa non risulta essere adatta per la ricerca di eventi in tempo reale, poiché la durata effettiva di un argomento è generalmente imprevedibile.

In particolare, un sistema in tempo reale deve essere capace di catturare eventi in maniera rapida, il quale produce una quantità enorme di tweet in un breve periodo di tempo. Tale comportamento può essere raggiunto adottando una finestra dinamica *W*, la cui dimensione dipende dalla funzione *Sigmoide*:

$$S(x) = c_1\left(1 - \frac{1}{1 + e^{-c_2(x - c_4)}}\right) + c_3$$

dove i parametri c_1, c_2, c_3, c_4 controllano rispettivamente l'intervallo dinamico, la pendenza, l'inclinazione e il centro della funzione *Sigmoide* [20]. Cosi, possono essere usate delle finestre piccole per poter riconoscere eventi in cui ci sono un numero elevato di tweet (es: c_1 = 20000 è la soglia per chiudere istantaneamente la finestra).

Come si può vedere nella Figura 3, il trend della curva cambia dopo 10 minuti (c_4=10) e più il tempo passa, meno tweet sono necessari per completare la finestra di ricerca. La pendenza c_2 è stata fissata a 0.3, mentre $c_3 = 200$ al fine di catturare almeno 200 tweet prima che inizi la fase di *Detection*.

Nonostante venga adottata una finestra dinamica, permette di adattarsi al comportamento del *detector* rispetto al volume attuale dei tweet. Per superare la limitazione dell'algoritmo *SFPM* nel trovare eventi inaspettati, viene mantenuto un set dinamico di termini che viene continuamente aggiornato, nel quale vengono

inseriti nuovi termini che rispecchiano gli argomenti di interesse che si stanno cercando, inoltre vengono anche eliminati i termini che non sono usati o risultano essere vecchi.

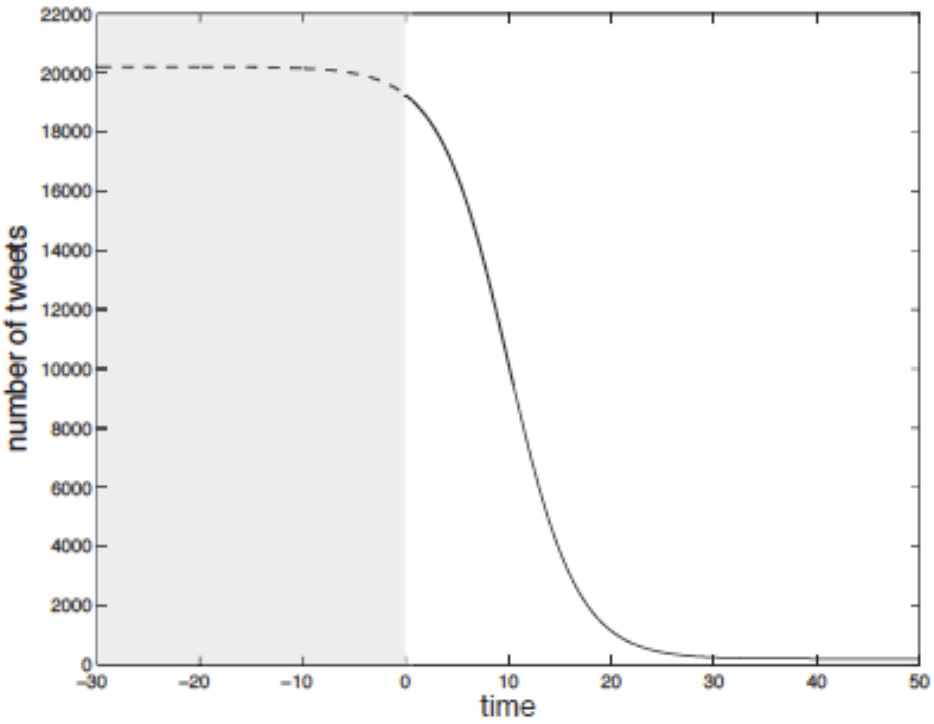


Fig. 3 La funzione Sigmoide che modella il comportamento della finestra di ricerca.

In maniera specifica, per l'*n-ma* finestra di *Detection* W_n si mantiene una lista L_n dei termini più rilevanti in W_n e un vettore di "score" I_{tn}, i cui valori rappresentano l'importanza di ogni termine t in W_n, calcolato come la radice quadrata del numero dei tweet dove il termine t si trova. I termini che hanno uno "score" al di sopra della media sono raggruppati e aggiunti ad L_n. Il ciclo di vita dei nuovi termini è implicitamente limitato ad una singola finestra, così da concentrarci su un evento specificato dal set di termini iniziali. L'intero comportamento del *Twitter Live Detection Framework* è sintetizzato in Figura 4.

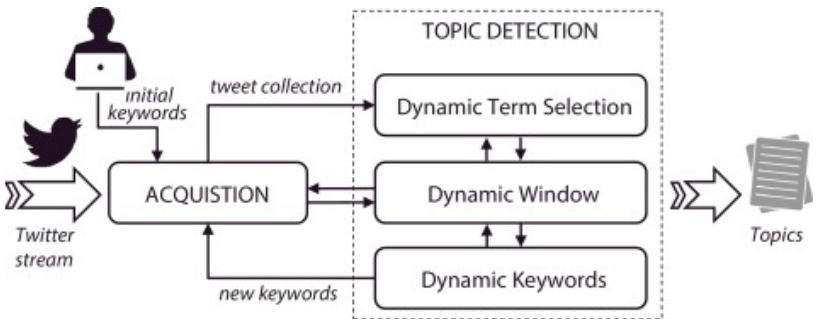


Fig. 4 Schema del comportamento del Twitter Live Detection Framework

3.4. Pseudocodice

Di seguito viene presentato lo pseudocodice dell'algoritmo *TLDF*. Questo algoritmo differisce dal SFPM, proprio al primo passo, quando viene chiamato il metodo per la selezione dei termini:

Twitter Live Detection Framework: SFPM(*C,K,b,c*)

1. *T = SFPM_TermSelection(C,K)*
2. Per **ogni** termine *t* in *T*:
    2.1. Calcola il vettore D_t
3. *Topics* = 0
4. Per **ogni** termine *t* in *T*:
    4.1. $S = t$
    4.2. $D_s = D_t$
    4.3. *ContinueExpandings = true*
    4.4. **Ripeti** finché *ContinueExpandings = true*
        4.4.1. $t = GetBestMatchingTerm(D_s, S, T)$
        4.4.2. $sim = ConsineSimilarity(D_s, D_t)$
        4.4.3. **Se** $sim > \theta(S)$
            4.4.3.1. $S = S \cup \hat{t}$
            4.4.3.2. $D_s = D_s + D_{\hat{t}}$
            4.4.3.3. **Per** $i = 1,...,n$
                4.4.3.3.1. **Se** $D_s(i) < |S|/2$
                    4.4.3.3.1.1. $D_s(i) = 0$
        4.4.4. **Altrimenti**
            4.4.4.1. *ContinueExpandings = false*
    4.5. $Topics = Topics \cup S$
5. *RemoveDuplicates(Topics)*

4. Il sistema proposto

L'obiettivo di questo progetto è quello di realizzare un sistema per tenere informato l'utente finale su un evento specifico, scaricando informazioni da Twitter, riuscendo a fornire più informazioni possibili in maniera dettagliata omettendo le informazioni non rilevanti o *spam*. Si è scelto di svilupparlo in *Java* e tramite lo sviluppo di alcune funzioni con le *Servlet* per poter essere in grado di affiancargli facilmente un'interfaccia grafica in modo da automatizzare tutto il sistema e renderlo disponibile a qualsiasi utente. Viene esposta adesso l'architettura complessiva di questo sistema.

Il sistema è stato suddiviso in quattro fasi distinte:

- *Ricerca*: questa è la parte di ricerca dei tweet e dei feed RSS, in cui sono state previste tutte le funzionalità e i metodi per poter inserire i dati per avviare il sistema di ricerca, scaricare i tweet e i feed RSS e successivamente salvarli;
- *Analisi*: questa è la parte di analisi dei dati precedentemente scaricati. Inoltre, si è volutamente suddivisa in due parti in cui: la prima prevede l'analisi dei dati con la finestra di ricerca statica, mentre nella seconda parte si procede all'analisi completa con la finestra di ricerca dinamica e quindi si è incluso un nuovo sistema di ricerca dinamico.
- *Post-Processing*: parte del sistema sviluppata per ridurre ulteriormente la presenza di argomenti non interessati dopo la fase di analisi.
- *Valutazione*: parte dedita alla valutazione dei dati estratti durante la precedente fase lasciando il pieno controllo all'utente finale.

È stata inoltre realizzata un'interfaccia grafica sia per poter collegare le fasi di ricerca, analisi, post-processing e valutazione, che per dare un sistema automatico al fine di estrarre informazioni il più possibile dettagliate su un evento generico, per l'utente finale.

4.1. Sistema di Ricerca

Il sistema di acquisizione proposto è stato composto da una parte dedicata al download dei tweet e da un'altra parte per il download dei feed RSS. Si è scelto di aggiungere anche i *feed* RSS per arricchire la ricerca. Entrambi vengono svolti in

parallelo ma con due thread distinti e separati. Per quanto riguarda questa prima fase di acquisizione, è stata realizzata una Servlet che riceve in ingresso i seguenti parametri:

- *keywords*: sono le chiavi di ricerca dei tweet e dei feed RSS, dove le virgole separano ogni singola chiave, in modo da dare come input più parole per affinare la ricerca;
- *fileName*: sarà il nome del file di salvataggio che verrà creato alla fine di ogni sessione di acquisizione;
- *langaugeTweet*: questo è il campo che rappresenta la lingua usata per filtrare i tweet e può anche essere lasciato vuoto, in questo caso scaricherà tweet scritti in qualsiasi lingua. In una fase successiva, verrano scartati i tweet che non sono in lingua inglese visto che per il sistema proposto è stato scelto preventivamente soltanto l'uso della lingua inglese;
- *appendTweet*: questo campo se impostato a *true* da la possibilità di continuare la sessione precedente appendendo i nuovi tweet nel file precedente, senza riscriverlo da capo;
- *indexStartFile*: nel caso in cui si volesse continuare la sessione precedente indica l'indice successivo del file;
- *DateStartRSS*: campo che indica l'inizio della sessione di acquisizione dei feed RSS che si vogliono scaricare;
- *minuteIntervalRSS*: intervallo di tempo che intercorre per la verifica della presenza di nuovi feed RSS;
- *RSS*: campo dove viene indicato il servizio RSS da utilizzare, questo campo risulta particolare in quanto deve essere scritto racchiuso dai tag "*RSS:[*" e "*]*" e all'interno vanno inserite le seguenti informazioni:
    - *URL*: indirizzo web dal quale scaricare i feed RSS;
    - *Nome*: il nome fornito dal servizio, il quale viene utilizzato come nome per la creazione e il salvataggio del file relativo ai risultati;
    - *appendRSS*: campo opzionale, come nel caso dei tweet, composto da *false* o *true* indica se continuare una sessione precedente oppure no;
    - *indexStartRSS*: come nel caso dei tweet, indica l'indice da cui iniziare per il salvataggio del file.

Una volta inseriti questi parametri attraverso l'interfaccia grafica e dopo aver fatto i relativi controlli di forma, viene avviato il processo di acquisizione e salvataggio dei dati realizzato attraverso il linguaggio di programmazione Java.

Di seguito, verrà spiegato nel dettaglio come vengono scaricati i Tweet, i Feed RSS, i metodi e i parametri che compongono la fase di salvataggio.

4.1.1. APIs Twitter

Twitter mette a disposizione agli sviluppatori le *API* (*Application Programming Interface*), insieme di procedure per poter accedere facilmente ai contenuti e alle procedure di un certo programma. La finalità delle API, generalmente, consiste nel fornire un livello di astrazione tra un servizio (livello inferiore) e il suo fruitore, che può essere a sua volta un altro servizio o un altro software.

Nel caso in esame, l'interesse è volto alla lettura e all'acquisizione dei tweet [21]. Gli strumenti che mette a disposizione Twitter sono molteplici, ognuno dei quali è stato sviluppato per svolgere un compito specifico. In questo elaborato, si è scelto di utilizzare le *Straming APIs* [22], le quali offrono la possibilità allo sviluppatore di accedere all'intero flusso globale dei tweet in tempo reale. Tramite la corretta implementazione di un client di streaming è possibile scaricare i tweet che pubblicano gli utenti o altri tipi di eventi che accadono all'interno della piattaforma. Inoltre, Twitter offre diverse modalità di accesso ai dati:

- *Flusso Pubblico*: informazioni pubbliche che fluiscono su Twitter. Utile a seguire specifici utenti o argomenti (ottimo per effettuare *data mining*).
- *Flusso dell'utente*: informazioni relative al singolo utente, contiene pressappoco tutti i dati corrispondenti alla vista del singolo utente nel suo account Twitter.
- *Flusso del sito*: è la versione multi-utente della precedente tipologia. Specifico per i server che devono connettersi a Twitter per conto di un gran numero di utenti.

A differenza di quanto accade per le *REST API* standard, connettersi alle Twitter *Streaming APIs* richiede l'instaurazione di una connessione *HTTP* persistente. Ciò influisce, chiaramente, su tutta l'architettura dell'applicazione che utilizza tale insieme di funzionalità.

Per tale motivo, infatti, il sistema è stato realizzato in Java tramite l'utilizzo della libreria *Twitter4j* [25], una libreria non ufficiale in Java che permette l'utilizzo delle *API* di Twitter. Con *Twitter4j*, si può facilmente integrare l'applicazione Java con i servizi di Twitter, quindi, permette di accedere allo streaming dei tweet con facilità utilizzando il linguaggio Java. Infine, è stato scelto il *flusso pubblico* come *Streaming API* per il sistema.

Le *API* di Twitter prevedono che tutte le richieste, ad esclusione di poche eccezioni, forniscano i parametri richiesti al fine di implementare un'autorizzazione *OAuth*. Perciò, è necessario procurarsi i valori di tali parametri. A tale scopo è indispensabile creare un'applicazione Twitter collegandosi al gestore applicazioni [23]. Cosi, si è creata un'applicazione a scopo di test ottenendo una *consumer key* e una *secret key* (o *consumer secret*), che sono indispensabili nel processo di autenticazione e autorizzazione delle *API* di Twitter in quanto, codificate, identificano univocamente l'applicazione.

Anche se le *API* di Twitter sono fruibili da chiunque, presentano alcune limitazioni nel loro utilizzo [24]. Le cause maggiori della chiusura della streaming sono: quando un utente stabilisce troppe connessioni con le stesse credenziali, quindi bisogna fare attenzione a non avviare due connessioni in parallelo con le stesse credenziali; se un utente ferma la lettura dei dati all'improvviso la connessione può essere persa; se un utente legge i dati troppo lentamente, quindi la coda dei messaggi diventa molto grande allora la connessione può essere persa. Queste piccole limitazioni vanno considerate durante la fase di elaborazione dei tweet e quindi, per il sistema di *Detection* da realizzare, per non incorrere nelle limitazioni appena discusse.

4.1.2. Feed RSS

RSS è uno dei formati più popolari per la distribuzione di contenuti *Web* ed è basato sulla stessa sintassi di *XML*, da cui ha ereditato la semplicità, l'estensibilità e la flessibilità [26]. *RSS* definisce una struttura adatta a contenere un insieme di notizie, ciascuna delle quali sarà composta da vari dati (nome autore, titolo, testo, etc.). Quando i fornitori stessi del servizio pubblicano delle notizie in formato RSS, la struttura viene automaticamente aggiornata con i nuovi i dati; visto che il formato è predefinito un qualunque lettore RSS potrà presentare in maniera omogenea notizie provenienti da più fonti diverse.

Per tale motivo, in parallelo all'acquisizione dei Tweet, si è realizzato un metodo in Java per poter scaricare i *feed RSS* dai siti *Web* che li mettono a disposizione, in modo da poter arricchire la ricerca di argomenti interessanti.

4.1.3. Salvataggio delle informazioni

Una volta avviata la fase di acquisizione dei tweet, ogni qualvolta si scarica un nuovo tweet, viene immediatamente salvato sul file. Per quanto riguarda gli *RSS*, non avendo un sistema come quello delle *API* di Twitter si è scelto di utilizzare un intervallo di tempo, in cui, il sistema controlla periodicamente la presenza di nuovi *feed RSS* aggiornati dal fornitore del servizio stesso.

Su ogni tweet scaricato si è scelto di salvare le seguenti informazioni, che risultano essere le più rilevanti al fine della costruzione del sistema:

- *id*: una campo numerico rappresentante la chiave identificativa del tweet all'interno del database di Twitter;
- *rt*: campo booleano che se è impostato a *true* il tweet in questione rappresenta un retweet, cioè una copia di un messaggio di un'altro utente, altrimenti non lo è;
- *text*: campo che contiene il testo del messaggio, nel caso di retweet questo campo inizia con la parola "RT:", rappresentando un'altra fonte di rumore;
- *lang*: il linguaggio del tweet, esso non è sempre veritiero in quanto alcuni utenti scrivono in altre lingue rispetto a quella dichiarata in fase di iscrizione;
- *uploadTime*: data e ora di pubblicazione del tweet;
- *uploader*: il nome dell'utente, ossia il nome impostato dall'utente e visualizzatile a tutti;
- *hashtag*: una stringa contenente gli hashtag definiti dall'utente nel corpo del messaggio, ogni hashtag è separato dalla virgola;
- *user_id*: campo numerico rappresentante una chiave univoca dell'utente nel database di Twitter;
- *user_location*: campo contenente la locazione dell'utente durante la fase di iscrizione a Twitter;
- *user_lang*: lingua impostata per l'account Twitter dell''utente;

- *utc_offset*: rappresentante lo scarto in secondi per il fuso orario impostato dall'utente rispetto allo Coordinated Universal Time (UTC);

Per ogni *feed* RSS scaricato, invece, vengono salvate le seguenti informazioni:

- *title*: è il titolo del feed RSS su cui viene effettuata la ricerca al fine di determinare l'attinenza del messaggio con le chiavi di ricerca definite dall'utente;
- *descrizione*: contenente una breve descrizione dell'evento, ma viene spesso usato per contenere piccole strutture html usate dai browser per conferire una struttura ed una formattazione ai testi;
- *autore*: contiene il nome dell'autore del messaggio;
- *pubDate*: data e ora di pubblicazione del feed RSS.
- *guid*: id del gruppo dei messaggi di cui fa riferimento il feed scaricato;
- *link*: è il link di appartenenza della notizia fornito dal servizio RSS;

Per il salvataggio dei file, sia per quanto riguarda i tweet che per i *feed* RSS, è stato scelto il formato *JSON* (*JavaScript Object Notation*). È un semplice formato per lo scambio di dati e la sua semplicità ne ha decretato un rapido utilizzo specialmente nella programmazione in AJAX (*Asynchronous JavaScript and XML*), utilizzato dallo stesso come alternativa ad *XML*. Il suo uso tramite JavaScript è particolarmente semplice, infatti l'interprete è in grado di eseguire il parsing tramite una semplice chiamata ad una funzione. Questo è uno dei motivi principali che lo ha reso molto popolare e anche il motivo per la quale si è scelto di utilizzarlo.

Per non creare file di grosse dimensioni, al fine di evitare che in fase di *Detection* si avessero problemi, si è scelto di creare un singolo file con dimensione massima di 100.000 tweet.

4.2. Sistema di Analisi

Si è scelto di suddividere in due parti distinte questa parte del sistema:

- *Detection*: fase di analisi dei dati scaricati con la finestra di ricerca statica, cioè si inserisce il file acquisito nella fase precedente e si effettua un'analisi su quei dati estraendo argomenti di interesse.

- *Live Detection*: sistema finale che prevede sia la fase di acquisizione che la fase di analisi, realizzata tramite la creazione di una funzione che controlla lo stop e l'avvio della fase di ricerca e la fase di Detection in base all'uso della finestra dinamica e quindi dell'aggiornamento delle nuove *keyword* di ricerca per effettuare ricerche sempre più mirate sull'argomento interessato.

La scelta di lasciare entrambe le due fasi è risultata utile per verificare la sostanziale differenza tra l'uso della finestra statica rispetto a quella dinamica con l'aggiornamento delle *keyword* di ricerca.

4.2.1. Detection

Il sistema di analisi statica proposto è stato realizzato implementando due funzioni principali.

Una prima fase in cui si prevede che l'utente tramite interfaccia grafica scelga su quale file *JSON*, precedentemente salvato durante la fase di acquisizione, fare l'analisi e l'intervallo di tempo su cui effettuare l'analisi. Per esempio, anche se il file di acquisizione ha scaricato tweet per la durata di 2 ore, l'utente può opportunamente decidere se effettuare l'analisi sull'intera durata del set oppure su un intervallo di tempo ben preciso.

Una seconda fase in cui sulla finestra temporale scelta viene applicato il processo di analisi. Una volta completata la fase di Detection ne verranno salvati i risultati su file XML separati, in cui verranno anche aggiunti gli eventuali *feed* RSS. Questo file XML conterrà tutti i Topic rilevanti trovati dall'analisi. Per ogni Topic trovato saranno salvate le seguenti informazioni:

- *id*: una stringa alfanumerica che identifica univocamente il Topic;
- score: il punteggio attribuito in base all'importanza del Topic;
- keywords: riporta la lista dei termini rilevanti che fanno parte di quel dato Topic, inoltre per ogni termine verrà salvato:
    - term: la parola estratta;
    - score: il punteggio attribuito per la sua presenza nel documento;
- relevantTweet: una lista dei tweet rilevanti agglomerati sotto lo stesso argomento e quindi assegnati allo stesso Topic, inoltre per ogni tweet ritenuto rilevante verrà salvato:
    - *id*: stringa numerica che identifica univocamente il tweet;

- *text*: il testo del tweet;
- *uploader*: il nome dell'utente che ha scritto il tweet;
- *uploadTime*: data e ora della pubblicazione del tweet;

Verrano anche salvate l'orario di inizio e fine della finestra temporale dell'analisi e le keyword che sono state immesse durante la fase di ricerca che si riferiscono all'attuale finestra.

Alla fine di questi due processi verra creato un ulteriore file XML che conterrà la lista dei file di sintesi creati durante la fase di analisi.

4.2.2. Live Detection

Il sistema di analisi in tempo reale proposto è stato realizzato aggiungendo e modificando alcune funzioni sviluppate per il sistema di analisi statica e di acquisizione precedentemente esposte.

Per primo, si è scelto di integrare la fase di acquisizione dei tweet e dei feed RSS al sistema di analisi attraverso lo sviluppo di una funzione *Controllore*, che collega la fase di acquisizione con quella di analisi aggiornando le keyword di ricerca e utilizzando la finestra di ricerca dinamica.

Innanzitutto il controllore avvia la fase di acquisizione con i parametri impostati dall'utente attraverso l'interfaccia grafica. Successivamente la funzione controllore, ad ogni minuto, verifica se la soglia temporale della finestra viene superata. Quindi, la soglia tiene conto del numero dei tweet scaricati fino a quel momento e del tempo trascorso dall'avvio del sistema di ricerca. Se questa soglia viene superata, il controllore ferma la fase di acquisizione e avvia il sistema di Detection consegnandogli l'insieme di tweet su cui effettuare l'analisi. Se la soglia non viene superata allora continua a scaricare i tweet. La soglia utilizzata è stata ampiamente descritta nel capitolo 2.

Una volta completata la fase di *Detection* e individuate le nuove keyword di ricerca, il sistema stesso avvisa il controllore che si occupa di aggiornare il filtro del sistema di acquisizione e di riavviare la ricerca con le nuove keyword per individuare ulteriori argomenti di interesse. Oltre questo sistema sono stati aggiunti altri metodi all'interfaccia grafica per poter arrestare il sistema e avviare la fase di valutazione.

Questo sistema, così strutturato, si adatta correttamente alle keyword immesse dall'utente quando avvia il sistema. Infatti si nota che quando vengono trovati molti tweet dal sistema di ricerca la finestra si riduce fino ad intervalli di tempo di 1 minuto, mentre se non trova numerosi tweet la finestra resta fissa su intervalli di tempo di 20 minuti. Il vantaggio maggiore si percepisce nell'aggiornamento automatico delle keyword. Infatti, oltre a trovare argomenti di interesse sempre più precisi, si riescono a trovare anche argomenti estranei alle keyword iniziali, ma che in quel dato istante risultano essere maggiormente frequenti per quel dato argomento.

Come risultato, da questo sistema di Live Detection, avremo una serie di Topic rappresentanti gli argomenti di interesse e per ognuno di essi la lista delle parole chiave utilizzate per la ricerca inerenti al Topic.

I dati estratti, come avveniva nella fase di Detection statica, vengono memorizzati in due file XML, uno dove vengono riportati tutti i Topic e le keyword estratte e l'altro che conterrà la lista dei file di sintesi creati durante la fase di Detection. Questi risultati saranno dati in input al sistema di post-processing, che verrà descritto nel paragrafo successivo.

4.3. Post-Processing

Per discriminare ulteriormente i tweet appartenenti ai Topic creati dal sistema di analisi è stato creato un ulteriore processo di post-processing strutturato in due fasi:

1. Fase preliminare in cui ad ogni Topic estratto, durante la fase di analisi, viene associato soltanto un tweet inerente all'argomento del Topic;
2. Fase aggiuntiva per discriminare ulteriormente la presenza di spam tra i tweet estratti durante la prima fase;

Durante la fase preliminare vengono esaminati tutti i Topic risultanti dalla fase di Detection e per ogni Topic si effettua il seguente processo:

1. Si estraggono le keyword e il loro relativo "score" (frequenza del termine all'interno del Topic);
2. Per ogni tweet rilevante appartenente al Topic, si estraggono tutti i termini del tweet e ognuno di essi si confronta con le keyword estratte al punto 1;

3. Per ogni termine che coincide con uno delle keyword estratte al punto 1, gli si aggiunge il relativo “score” ;
4. Viene costruito un Evento rappresentato dal tweet e da un valore che sarà la somma di tutti gli “score” dei singoli termini trovati all’interno del tweet;

Quindi, ogni Topic sarà composto da una lista di tweet con il loro valore di interesse, maggiore sarà questo valore maggiore sarà l’importanza del tweet all’interno del Topic.

La lista dei tweet viene ordinata in ordine crescente in base al loro valore di interesse e viene preso il tweet con il valore di interesse maggiore. Così, viene estratto presumibilmente il tweet che più rappresenta il Topic in esame. Da questa prima fase preliminare si ha una sostanziale riduzione del numero dei tweet rispetto al numero iniziale estratto durante la fase di Detection.

Conclusa questa prima fase preliminare, si passa alla seconda fase denominata *Topic Refinement*. La *Topic Refinement* procede nel confrontare le keyword iniziali della ricerca con ogni tweet estratto dalla prima fase. Cosi, si creano due liste distinte, una contenente i tweet che presentano più di due termini in comune con le keyword iniziali, la seconda lista sarà composta dai tweet che presentano un solo termine in comune con le keyword iniziali:

- Lista dei tweet con due termini o più uguali alle chiavi di ricerca;
- Lista dei tweet con un solo termine uguale alle chiavi di ricerca;

Una volta completate le due liste, l’algoritmo procede nel creare un vocabolario con tutti i termini present nei tweet della lista con almeno due o più termini uguali alle chiavi di ricerca. Il vocabolario è cosi costruito:

- *termine*: la parola vera e propria;
- *frequenza*: numero di volte che il termine compare sia all’interno del tweet, che in tutti i tweet della lista con almeno due termini uguali alle chiavi di ricerca;

Come risultato avremo, all’interno del vocabolario una lista dei termini con la loro effettiva frequenza di co-occorrenza nell’insieme dei tweet in esame. Completato il vocabolario, si confronterà ogni termine presente nel vocabolario con ogni termine estratto dai tweet appartenenti alla lista dei tweet con un solo termine in comune alle chiavi di ricerca iniziali. Quindi, verrà creata una lista di tweet

(meno buoni) con un certo "score" composto dalla somma delle frequenze dei singoli termini che compaiono nel tweet in esame e che sono presenti nel vocabolario appena costruito.

Infine, si filtra la lista dei tweet risultante con un certo valore di soglia, se il tweet supera la soglia viene preso altrimenti viene scartato. I tweet che superano la soglia verrano aggiunti alla prima lista dei tweet che hanno almeno due termini in comune con le chiavi di ricerca e vanno a comporre la lista finale dei tweet totali. Fatto ciò, i tweet vengono ulteriormente scremati andando ad eliminare quelli che risultano essere scritti in maniera identica e infine vengono salvati in un file XML, memorizzato per ogni tweet le seguenti informazioni:

- *id*: valore numerico assegnato univocamente al tweet;
- *date*: data e orario di pubblicazione del tweet;
- *text*: testo del tweet;

Si può notare che, applicando questo processo, con una soglia leggermente superiore alla media della frequenza dei termini si riesce a ridurre notevolmente il numero dei tweet a discapito della rimozione di qualche tweet interessante.

4.4. Sistema di Valutazione

Come esplicitamente descritto in [1] e [13], in letteratura esistono diversi criteri per valutare i risultati prodotti da un sistema simile. Si prevede in [13] un metodo automatico dove si confrontano le keyword utilizzate per ogni finestra di ricerca con le liste dei termini presenti in ogni tweet rilevante assegnato al Topic. Così, viene indicata l'appartenenza del Topic all'evento di interesse, ma considerato che nel sistema appena presentato nella fase di post-processing si utilizza un metodo simile per eliminare ulteriormente tweet che non fanno parte dell'evento ricercato, questo metodo non è stato utilizzato. Quindi, si è scelto di utilizzare un metodo manuale per annotare ogni singolo tweet e calcolare una valore di precisione medio sull'insieme totale dei tweet estratti. Questo, perché stiamo valutando un sistema creato per individuare eventi importanti in un flusso di dati molto rumoroso e non strutturato come Twitter.

Con l'ausilio dell'interfaccia grafica è stato realizzato un sistema che visualizza le informazioni contenute nel file di sintesi estratto dalla fase di post-

processing per permettere all'utente finale di poter discriminare ogni singolo tweet assegnandogli i seguenti valori:

- *Chiarezza*: quanto il tweet in esame è scritto correttamente, un valore che va da 1 a 10;
- *Pertinenza*: quanto il tweet in esame è ritenuto pertinente con l'argomento in esame, un valore che va da 1 a 10;
- Evento: se il tweet in esame rappresenta un argomento interessante;
- Duplicato: se il tweet in esame è già stato scritto anche con parole simili in un'altro tweet trovato precedentemente;
- Spam: se il tweet in esame non risulta essere inerente al contesto o non riguarda nessun evento;

Quindi, l'utente finale annoterà manualmente per ogni tweet questi valori al fine di produrre un file di sintesi dove verrà salvata tutta la valutazione. Il file di sintesi creato in XML conterrà le seguenti informazioni:

- Il numero di Topic etichettati come Evento;
- Il numero di Topic etichettati come Duplicato;
- Il numero di Topic etichettati come Spam;
- Tutti i tweet con la categoria assegnata;
- Il valore di chiarezza medio;
- Il valore di pertinenza medio;

I parametri di *chiarezza* e *pertinenza* estratti risultano utili nel capire in che modo i risultati prodotti dalla fase di analisi e di post-processing incidono sul risultato finale. Mentre per quanto riguarda la distinzione tra Evento, Duplicato e Spam, risulta utile proprio perché il sistema è stato creato per avvisare l'utente finale su un argomento ben preciso, quindi diventa utile capire quanti di questi risultino essere notizie importanti da poter essere notificate rispetto alle notizie non importanti.

4.5. Interfaccia Grafica

Per permettere all'utente finale di utilizzare in maniera semplice, intuitiva e automatica il sistema realizzato, è stata affiancata al sistema un'applicazione web attraverso la creazione di un'architettura client-server.

Attraverso l'uso di tecnologie come *HTML*, *CSS*, *JavaScript*, *JQUERY*, *AJAX*, *XML*, *XSL*, XHTML, *JSON, Servlet,* si è realizzato un sistema di visualizzazione per il sistema di ricerca, per il sistema di analisi, per il post-processing, per la valutazione e la visualizzazione dei risultati. Importante sottolineare che tutto il sistema è stato creato in locale attraverso l'ambiente di sviluppo *Eclipse* con la configurazione di un server in locale con *Apache Tomcat* (application server open source) e per affiancargli il database in MySQL si è utilizzato XAMPP.

Fig. 5 Interfaccia grafica della pagina per accedere al sistema, index.html.

Inoltre, per rendere ancora più facilmente accessibile l'interfaccia grafica si è utilizzato *Bootstrap* [27]. Sviluppato da Twitter, *Bootstrap* è una raccolta di strumenti *open source* per la creazione di siti e applicazioni Web. Contiene modelli di progettazione basati su *HTML* e *CSS*, sia per la struttura che per le varie componenti dell'interfaccia come moduli, pulsanti e navigazione, cosi come alcune estensioni opzionali di JavaScript. A partire dalla versione 3.0, *Bootstrap* ha adottato il *responsive design* come impostazione predefinita, ciò significa che il

layout delle pagine web si regola dinamicamente tenendo conto delle caratteristiche del dispositivo utilizzato sia esso desktop, tablet o smartphone.

Il sistema sviluppato è stato composto da una pagina principale, in cui l'utente per accedere al sistema deve effettuare il login (vedi figura 5). Si è creato appositamente un database in MySQL dove vengono immessi il nome utente e la password dell'utente a cui vengono dati i permessi per accedere al sistema. In fase di sviluppo, si è scelto appositamente di non prevedere momentaneamente una fase di registrazione.

Una volta effettuato l'accesso l'utente si trova all'interno dell'intero sistema e può accedere facilmente a tutte le sue fasi.

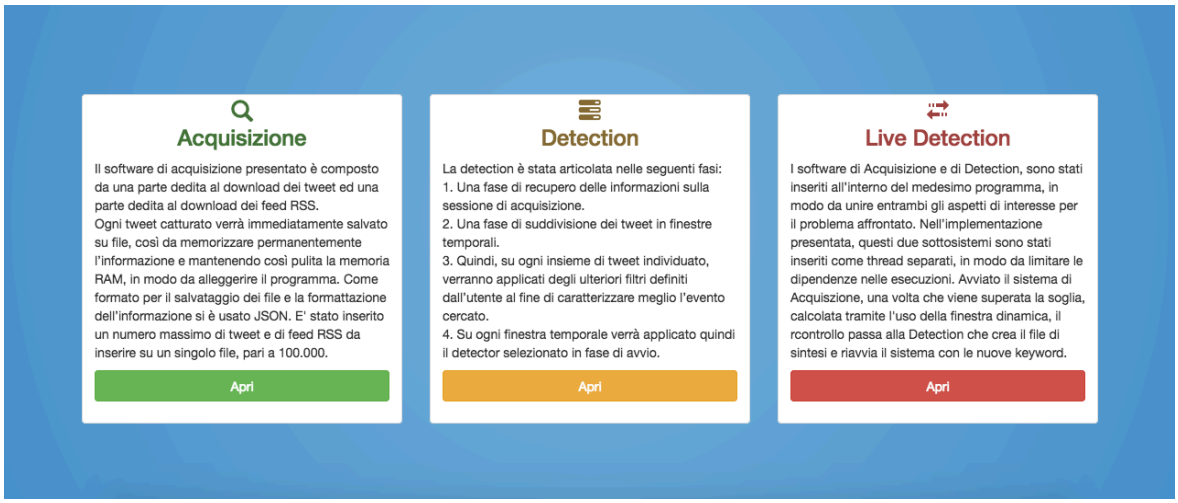


Fig. 6 Interfaccia grafica della pagina principale per l'accesso alle singole fasi del sistema, home.html.

Come mostrato in Figura 6, l'interfaccia grafica presenta all'utente le tre tipologie del sistema già discusso precedentemente. Quindi, l'utente può decidere se accedere al sistema di *Acquisizione*, di *Detection* o di *Live Detection*. Inoltre è stata sviluppata un'interfaccia grafica per il file di sintesi prodotto dall'analisi, per il file di sintesi prodotto dalla fase di post-processing e per la valutazione affidata completamente all'utente finale, quindi per la visualizzazione dei risultati ottenuti.

4.5.1. Sistema di Acquisizione

Per la realizzazione del sistema di *Acquisizione*, si è scelto di fornire all'utente finale un'interfaccia intuitiva dove potesse facilmente svolgere i seguenti compiti:

- Inserire le informazioni per configurare la sessione di Acquisizione;

- Visualizzare le informazioni inserite;
- Avviare, controllare e arrestare il sistema di Acquisizione.
- Visualizzare i tweet scaricati dal sistema di Acquisizione;

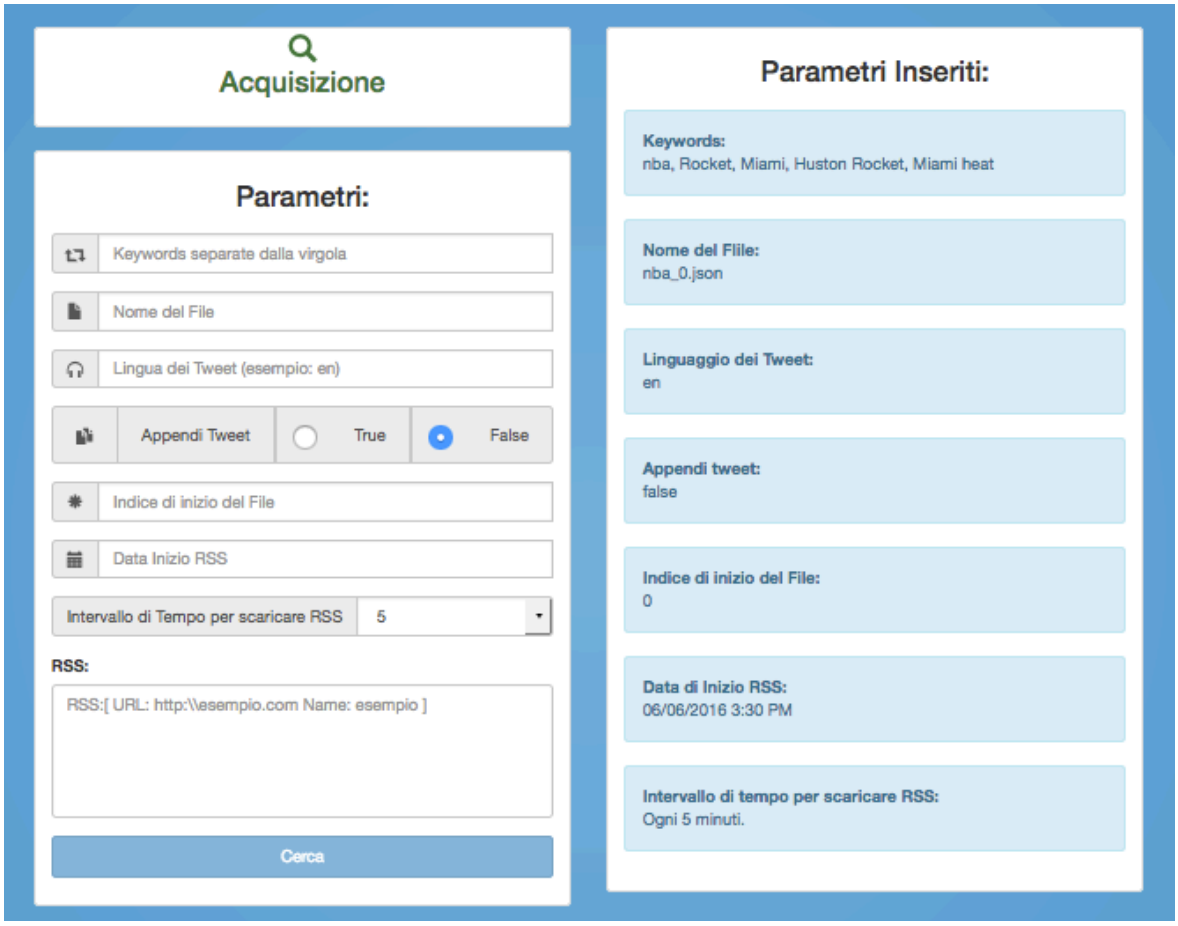


Fig. 7 Sistema di Acquisizione, acquisizione.html. A sinistra troviamo i campi relativi all'inserimento delle informazioni. A destra troviamo la sintesi dei dati inseriti.

Come mostrato in Figura 7, nella sezione riservata all'inserimento dei dati, vi sono diversi campi *input* per ogni valore che va a configurare la sessione di Acquisizione. Sono stati inseriti tutti i controlli di forma attraverso l'uso della libreria *JQuery*, quindi se l'utente dovesse erroneamente inserire informazioni sbagliate la ricerca non verrebbe avviata. Inoltre è stata aggiunta una sezione di riepilogo dei dati immessi, in quanto risulta comodo poterli verificare per poi poter fare una ricerca diversa o modificare alcune delle informazioni precedentemente utilizzate.

Quando viene avviato il sistema di Acquisizione si passa ad un'altra finestra, vedi Figura 8, in cui viene avviata la sessione di streaming nel sistema di Twitter e

contestualmente anche il download dei *feed* RSS se inseriti. Innanzitutto, il sistema prevede un controllo sull'esistenza del file, cioè prima di avviare lo streaming il sistema verifica se il nome del file inserito dall'utente è già presente nel sistema, se il file è presente chiede all'utente se intende procedere nel sovrascrivere il file o no. Successivamente, si procede all'acquisizione in cui sono stati forniti all'utente alcuni tasti per interagire con il sistema:

- *Tweet*: restituisce il numero dei tweet scaricati fino a quel momento;
- *RSS*: restituisce il numero dei feed RSS scaricati fino a quel momento;
- *Stop*: arresta la sessione di Acquisizione;
- *Close*: chiude la finestra;

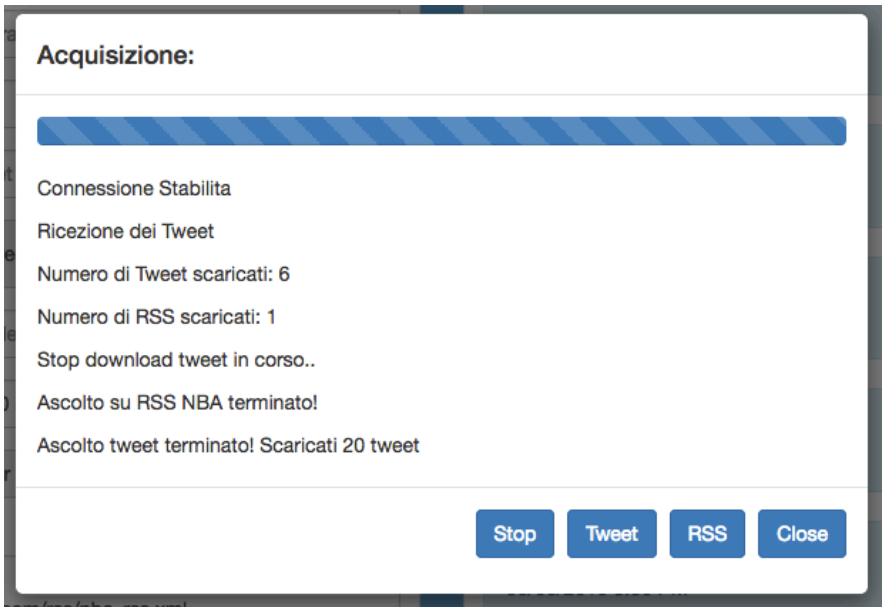


Fig. 8 Esempio di una sessione di Acquisizione e l'uso dei tasti.

Infine, è stata realizzata una piccola sezione dove poter visualizzare l'output dei tweet e dei feed RSS scaricati, cioè i file *JSON* memorizzati nel sistema. Si è scelto di visualizzare soltanto il nome utente e il testo (vedi figura 9).

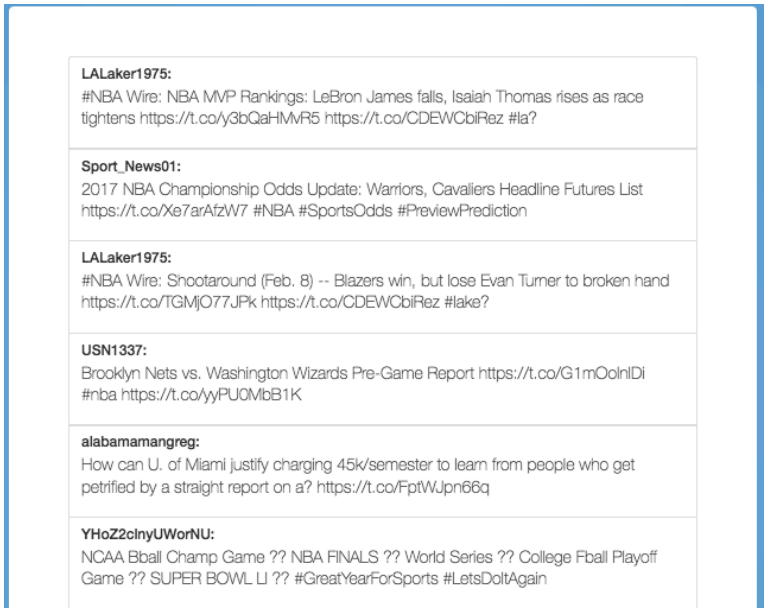


Fig. 9 Esempio visualizzazione tweet e feed RSS scaricati da una sessione di Acquisizione.

4.5.2. Sistema di Detection

Fig. 10 Sistema di Detection, detection.html. A sinistra troviamo la scelta del file di Acquisizione e il campo relativo alla scelta dell'intervallo di tempo.

Per la realizzazione del sistema di Detection, si è scelto di fornire all'utente finale un'interfaccia intuitiva dove potesse facilmente svolgere i seguenti compiti:

- Selezionare il file di acquisizione desiderato effettuato tramite il sistema precedentemente descritto. Inoltre, è stato previsto il controllo in cui se non

è presente alcun file nel sistema allora l'utente non può effettuare la Detection e viene rimandato nel sistema di Acquiszione;

- Scegliere l'intervallo di tempo dei dati scaricati su cui effettuare l'analisi;
- Avviare, controllare e attendere l'arresto della Detection;

Come mostrato in Figura 10, si può liberamente scegliere il file e l'intervallo di tempo su cui effettuare l'analisi. Quando viene avviata la Detection si passa ad un'altra finestra, vedi Figura 11, in cui l'utente può facilmente leggere le informazioni riguardanti la sessione di Detection, il caricamento del file JSON, l'avvio e il completamento della Detection.

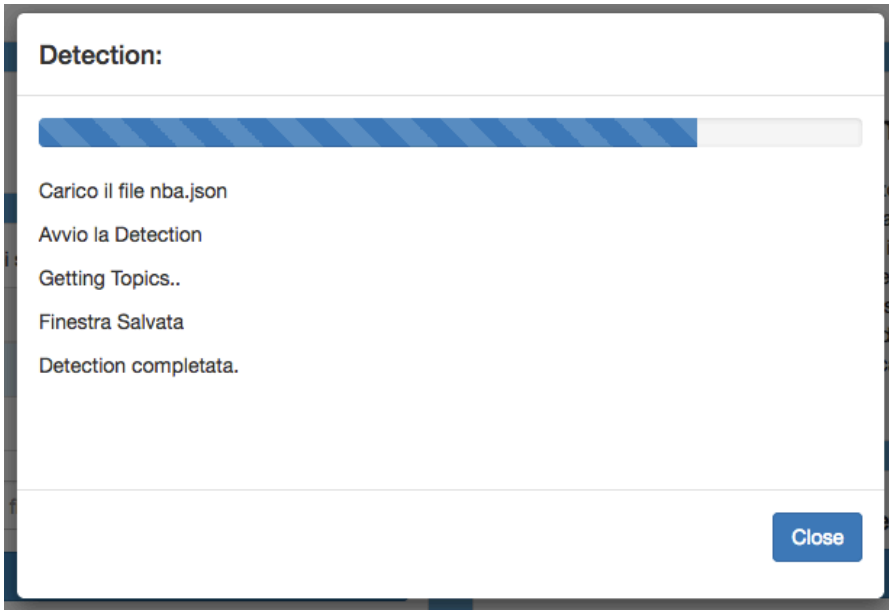


Fig. 11 Esempio di una sessione di Detection.

Infine, completata la Detection, i file di sintesi vengono memorizzati in un file XML e si può facilmente accedere ai risultati tramite interfaccia grafica (vedi Figura 12).

Fig. 12 Modulo aggiunto alla fine della Detection per accedere ai risultati.

4.5.3. Sistema di Live Detection

Per la realizzazione del sistema di *Live Detection*, si è semplicemente incorporata la parte relativa alla fase di *Acquisizione* con la parte relativa alla fase di *Detection*. Quindi si è lasciata l'interfaccia grafica già precedentemente sviluppata per l'acquisizione, vedi Figura 13.

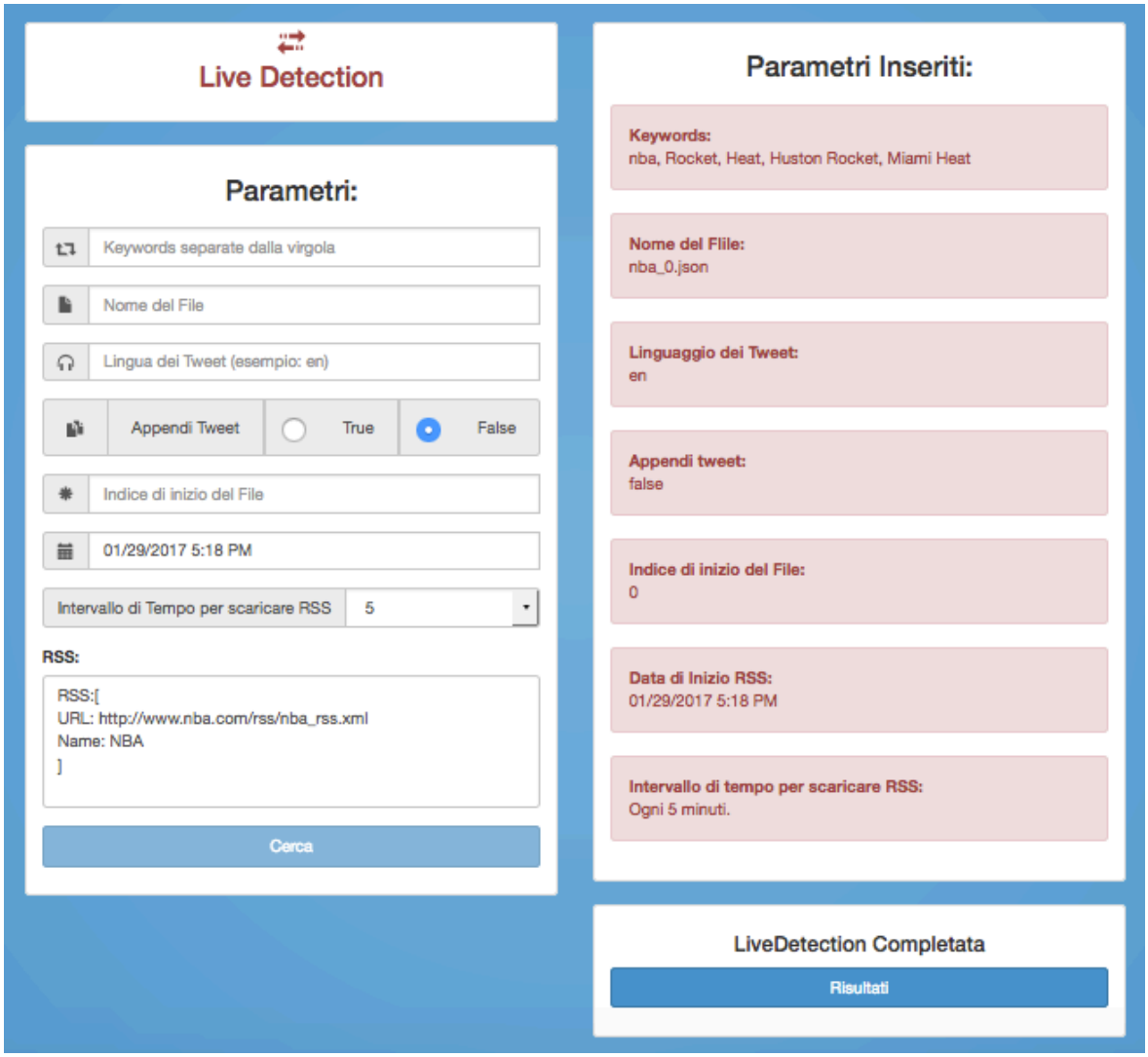


Fig. 13 Sistema di Live Detection, livedetection.html. A sinistra troviamo i campi relativi all'inserimento delle informazioni. A destra troviamo la sintesi dei dati inseriti in basso a destra troviamo il pulsante che porta ai risultati.

Le uniche modifiche apportate sono state sulla finestra di visulizzazione, mostrata in Figura 14, quando viene avviato il sistema di Live Detection. Infatti, sono stati aggiunti alcuni tasti per includere le funzioni innovative del sistema di Live Detection con l'uso della finestra dinamica:

- *Last Detection*: comunica all'utente l'orario in cui è stata effettuata l'ultima Detection e quindi l'ultimo aggiornamento delle keyword per riavviare il sistema di Acquisione;
- *Next Threshold*: comunica all'utente la soglia che deve essere superata per arrestare l'acquisizione e avviare la prossima *Detection*;

Inoltre, come si può notare in Figura 14, quando l'utente clicca sul tasto tweet viene avvisato non solo del numero dei tweet totali ma anche del numero dei tweet dell'attuale finestra.

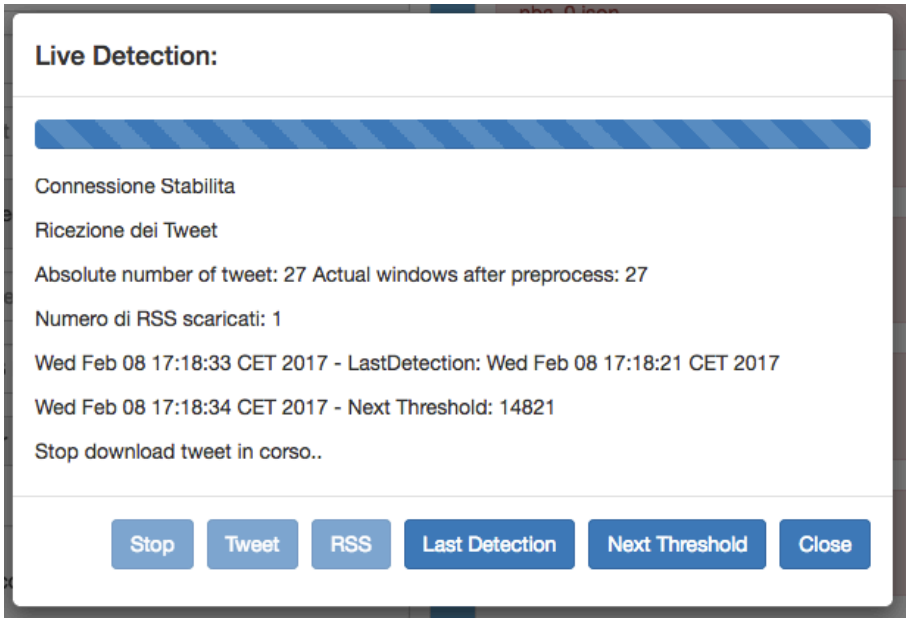


Fig. 14 Esempio di una sessione di Live Detection.

4.5.4. Visualizzazione dei risultati della Live Detection

Per permettere all'utente di poter accedere facilmente ai dati estratti dall'elaborazione del sistema e considerato che i file di sintesi vengono salvati in XML, attraverso l'uso di XSL e quindi di XHTML è stato facile rappresentarli graficamente. Sono state previste due pagine per poter rappresentare i dati.

Una pagina principale, vedi Figura 15, rappresentante il file main.xml dove in sequenza vengono riportati tutti i file di sintesi creati durante la fase di Detection. Per ogni file vengono visualizzate le seguenti informazioni:

- *FileName*: il nome del file di sintesi della Detection;
- *StartTime*: data e ora di inizio;
- *EndTime*: data e ora di fine;

- *topicsNumber*: numero dei Topic realizzati;
- *numberTweets*: numero di Tweet scaricati;

FileName	StartTime	EndTime	topicsNumber	numberTweets	ProgressBar
./output/nba/nba_method(5)_47.xml	Wed Jan 18 01:14:52 CET 2017	Wed Jan 18 01:16:53 CET 2017	14	15884	
./output/nba/nba_method(5)_48.xml	Wed Jan 18 01:16:53 CET 2017	Wed Jan 18 01:18:54 CET 2017	3	15629	
./output/nba/nba_method(5)_49.xml	Wed Jan 18 01:18:54 CET 2017	Wed Jan 18 01:20:55 CET 2017	7	20148	
./output/nba/nba_method(5)_50.xml	Wed Jan 18 01:20:55 CET 2017	Wed Jan 18 01:22:56 CET 2017	15	20040	
./output/nba/nba_method(5)_51.xml	Wed Jan 18 01:22:56 CET 2017	Wed Jan 18 01:23:58 CET 2017	4	20639	
./output/nba/nba_method(5)_52.xml	Wed Jan 18 01:23:58 CET 2017	Wed Jan 18 01:24:59 CET 2017	7	27371	
./output/nba/nba_method(5)_53.xml	Wed Jan 18 01:24:59 CET 2017	Wed Jan 18 01:27:00 CET 2017	5	19103	
./output/nba/nba_method(5)_54.xml	Wed Jan 18 01:27:00 CET 2017	Wed Jan 18 01:29:01 CET 2017	7	19659	
./output/nba/nba_method(5)_55.xml	Wed Jan 18 01:29:01 CET 2017	Wed Jan 18 01:31:03 CET 2017	8	23068	
./output/nba/nba_method(5)_56.xml	Wed Jan 18 01:31:03 CET 2017	Wed Jan 18 01:33:05 CET 2017	5	17781	
./output/nba/nba_method(5)_57.xml	Wed Jan 18 01:33:05 CET 2017	Wed Jan 18 01:35:06 CET 2017	9	21691	
./output/nba/nba_method(5)_58.xml	Wed Jan 18 01:35:06 CET 2017	Wed Jan 18 01:37:07 CET 2017	8	22180	
./output/nba/nba_method(5)_59.xml	Wed Jan 18 01:37:07 CET 2017	Wed Jan 18 01:39:08 CET 2017	4	19491	
./output/nba/nba_method(5)_60.xml	Wed Jan 18 01:39:08 CET 2017	Wed Jan 18 01:40:09 CET 2017	5	31040	
./output/nba/nba_method(5)_61.xml	Wed Jan 18 01:40:09 CET 2017	Wed Jan 18 01:41:10 CET 2017	4	21822	

Fig. 15 Visualizzazione grafica del file main.xml inserita nella pagina risultati.html.

Una seconda pagina, vedi Figura 16, a cui si può accedere cliccando ogni singolo file presente nella pagina precedente. Quindi, si è creata una semplice interfaccia per poter visualizzare i dati salvati nel file di sintesi in formato XML.

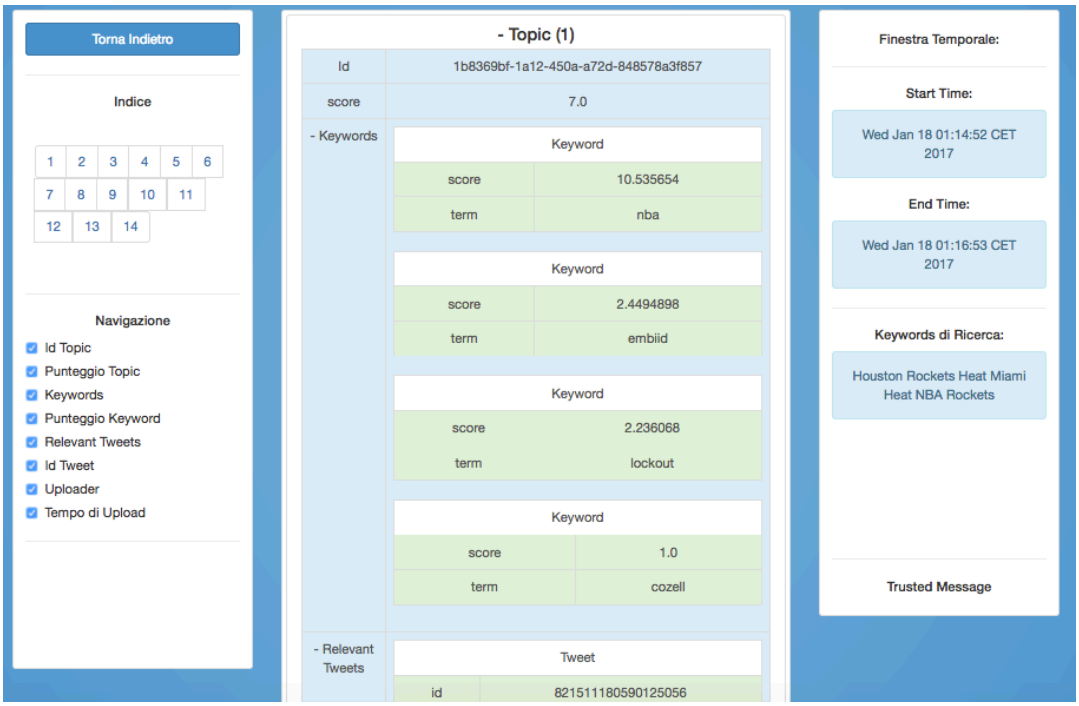
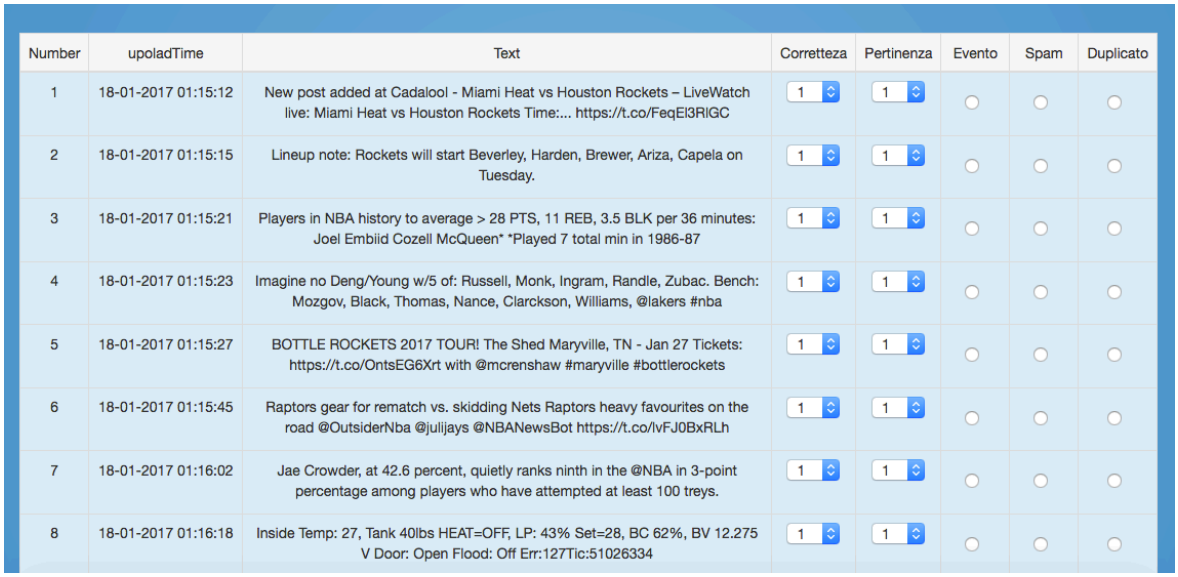


Fig. 16 Visualizzazione grafica di uno dei file di sintesi risultati dalla Detection.

Sulla sinistra, è stato creato un modulo di navigazione in cui l'utente può scegliere quale Topic e quali informazioni su di esso visualizzare. Sulla destra la sintesi della Detection di quel dato file, riportando orario di inizio e fine e le keyword di ricerca. Infine, nella parte centrale viene riportata l'informazione di ogni Topic trovato durante la fase di analisi.

4.5.5. Visualizzazione del Post-Processing e della Valutazione

Elaborata la fase di Post-Processing, come spiegato nella sezione precedente, i risultati vengono salvati in un file XML. Come primo passo, si è realizzata un'interfaccia grafica per poter visualizzare i risultati ottenuti. In aggiunta, per permettere all'utente di poter valutare i risultati, come descritto nel sistema di valutazione, si è realizzato un modulo che permettesse facilmente l'inserimento dei valori di *chiarezza*, *pertinenza*, *evento*, *spam* e *duplicato* già presentati.



Number	upoladTime	Text	Correttezza	Pertinenza	Evento	Spam	Duplicato
1	18-01-2017 01:15:12	New post added at Cadalool - Miami Heat vs Houston Rockets – LiveWatch live: Miami Heat vs Houston Rockets Time:... https://t.co/FeqEl3RlGC	1	1			
2	18-01-2017 01:15:15	Lineup note: Rockets will start Beverley, Harden, Brewer, Ariza, Capela on Tuesday.	1	1			
3	18-01-2017 01:15:21	Players in NBA history to average > 28 PTS, 11 REB, 3.5 BLK per 36 minutes: Joel Embiid Cozell McQueen* *Played 7 total min in 1986-87	1	1			
4	18-01-2017 01:15:23	Imagine no Deng/Young w/5 of: Russell, Monk, Ingram, Randle, Zubac. Bench: Mozgov, Black, Thomas, Nance, Clarckson, Williams, @lakers #nba	1	1			
5	18-01-2017 01:15:27	BOTTLE ROCKETS 2017 TOUR! The Shed Maryville, TN - Jan 27 Tickets: https://t.co/OntsEG6Xrt with @mcrenshaw #maryville #bottlerockets	1	1			
6	18-01-2017 01:15:45	Raptors gear for rematch vs. skidding Nets Raptors heavy favourites on the road @OutsiderNba @julijays @NBANewsBot https://t.co/lvFJ0BxRLh	1	1			
7	18-01-2017 01:16:02	Jae Crowder, at 42.6 percent, quietly ranks ninth in the @NBA in 3-point percentage among players who have attempted at least 100 treys.	1	1			
8	18-01-2017 01:16:18	Inside Temp: 27, Tank 40lbs HEAT=OFF, LP: 43% Set=28, BC 62%, BV 12.275 V Door: Open Flood: Off Err:127Tic:51026334	1	1			

Fig. 17 Visualizzazione grafica di uno dei file di sintesi risultati dalla fase di Post-Processing, valutazione.html.

Infine, dopo aver manualmente inserito tutti i campi, l'utente può avviare la fase di Valutazione e visualizzare i risultati sperimentali.

In questa ultima pagina del sistema, viene visualizzata la sintesi del sistema di Valutazione con i relativi grafici, come mostrato in Figura 18 e in Figura 19.

Il sistema di visualizzazione realizzato permette all'utente di poter visualizzare facilmente quanti sono i tweet interessanti o quanti sono duplicati o spam. Infine si è scelto di rappresentarli anche graficamente attraverso l'utilizzo della libreria *Chart.js* [28].

Fig. 18 Pagina dei risultati sperimentali, risultati-sperimentali.html. A sinistra troviamo il sistema di navigazione. A Destra i tweet raggruppati per Evento, Spam, Duplicato.

Chart.js è una libreria JavaScript che supporta la generazione di ben 6 tipi di grafici differenti dall'impatto estremamente accattivante. La libreria può essere scaricata gratuitamente dal repository ufficiale su *GitHub*. Il disegno del grafico si avvale del Canvas di HTML5 supportato dalla maggior parte dei browser moderni e risponde anche alle interazioni dell'utente adattandosi in modo "responsive" al device su cui viene prodotto. Tutte queste funzionalità sono racchiuse in uno script di piccole dimensioni (11 Kb circa), che possono essere ridotte ulteriormente se si usano solo alcuni dei tipi di grafici a disposizione.

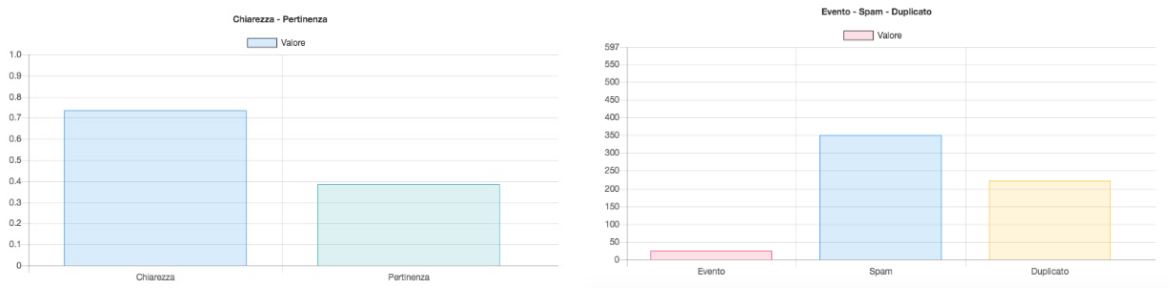


Fig. 19 Esempio visualizzazione dei grafici di Chart.js.

5. Risultati Sperimentali

Al fine di produrre i risultati sperimentali, si è scelto in fase di sperimentazione di elaborare come tipologia di dataset le partite di basket Americano più precisamente di *NBA* (*National Basket Assosation),* per la frequenza giornaliera delle partite e per la semplicità offerta nel poter estrarre argomenti di interesse al fine di avvisare l'utente finale. Inoltre, si è scelta questa tipologia di dataset anche per sfruttare il *NER* e quindi per l'obbligo d'uso della lingua inglese.

Durante la fase di studio dell'algoritmo *TLDF* e dei algoritmi di Post-Processing sviluppati sono stati analizzati diversi eventi, ed in questo elaborato verranno presentati i risultati relativi a tre incontri. Quindi, sono stati utilizzati i seguenti dataset:

- La partita di NBA tra *Huston Rockets* e *Miami Heat*. Tenutasi all'*AmericanAirlines Arena* giorno mercoledì 18 gennaio ore 01:30:00 (orario italiano), vinta da *Miami Heat*. È stato catturato il periodo che va dalle ore 01:14:52 alle ore 04:30:12;
- La partita di NBA tra *Toronto Raptors* e *Orlando Magic*. Tenutasi all'*Amway Center* giorno sabato 4 febbraio ore 01:00:00 (orario italiano), vinta da *Orlando Magic*. È stato catturato un periodo che va dalle ore 00:01:23 alle ore 03:55:27.
- La partita di NBA tra *Cleveland Cavaliers* e *Minnesota Timberwolves*. Tenutasi al *Target Center* mercoledì 15 febbraio ore 02:00:00 (orario italiano), vinta da *Cleveland Cavaliers*. È stato catturato un periodo che va dalle ore 01:44:44 alle ore 04:31:40.

5.1. Criteri di confronto proposti

Sono stati definiti tre criteri di valutazione: *chiarezza*, *pertinenza* e la distinzione tra *Evento*, *Duplicato* e *Spam*. Nel complesso il loro compito è di evidenziare se i dati estratti durante la fase di *Acquisizione* e di *Live Detection* risultano apprezzabili per poter distinguere eventi di interesse. Ne daremo una breve panoramica:

- *Chiarezza*: questo valore rappresenta la percentuale dei tweet che risultano essere scritti in maniera corretta da parte dell'utente andando a considerare la totalità dei tweet scaricati. Verrà cosi calcolato:

$$chiar(f) = \frac{\sum_{i=1}^{N} chiar_i}{chiar_{tot}}$$

Dove, N è il numero totale dei tweet, $corr_i$ è il valore di *chiarezza* assegnato al singolo tweet, che è un valore che va da 1 a 10 e $corr_{tot}$ è il valore massimo che può raggiungere la *chiarezza*, cioè $chiar_{tot} = 10 \times N$. Valori prossimi ad 1 indicano la bontà nell'individuare eventi scritti correttamente.

- *Pertinenza*: questo valore rappresenta la percentuale dei tweet che risultano essere all'interno del contesto delle chiavi di ricerca e quindi argomenti di interesse. Questo valore viene misurato su ogni singolo tweet e confrontato con la totalità dei tweet scaricati. Verrà così calcolato:

$$pert(t) = \frac{\sum_{i=1}^{N} pert_i}{pert_{tot}}$$

Dove, N è il numero totale dei tweet, $pert_i$ è il valore di *pertinenza* assegnato al singolo tweet, che è un valore che va da 1 a 10 e $pert_{tot}$ è il valore massimo che può raggiungere la *pertinenza*, cioè $pert_{tot} = 10 \times N$. Valori prossimi ad 1 indicano la bontà nell'individuare argomenti interessanti.

- *Evento, Spam, Duplicato*: questi tre valori rappresentano la totalità dei tweet distinti per ogni singolo tipo di tweet, nella totalità rappresentano un unico criterio di valutazione. *Evento* è il numero totale dei tweet che vengono selezionati come argomento interessante per il fine della ricerca; *Spam* è il numero totale dei tweet che vengono evidenziati come pubblicità o qualcosa non inerente al contesto della ricerca; *Duplicato* è il numero totale dei tweet che vengono evidenziati come tweet che si ripetono (scritti con parole diverse), sia tra gli eventi sia tra lo Spam.

5.2. Setup sperimentale della Topic Refinement

Prima di mettere a confronto l'algoritmo di *TLDF* presente in letteratura con i metodi presentati in questo elaborato è stata effettuata una fase preliminare di valutazione sulla *Topic Refinement,* per poter scegliere la soglia migliore al fine di estrarre più tweet che rappresentassero argomenti interessanti andando a scartare il più possibile la presenza di informazioni non rilevanti.

È stata condotta una sperimentazione sui dataset prodotti per poter valutare quale soglia assegnare all'algoritmo di Topic Refinement che meglio discrimina il set iniziale di tweet.

Per tale motivo, per poter filtrare i tweet dati in ingresso all'algoritmo di Topic Refinement si è proceduto a testare un valore di soglia θ compreso tra 0 e 1. Considerando che $\theta = 0$ significa che nessun tweet viene filtrato, quindi la lista dei tweet finale resterà immutata, mentre con $\theta = 1$ si prende in considerazione il valore massimo portando all'eliminazione di argomenti interessanti.

Quindi, come descritto nella fase di elaborazione del sistema, la Topic Refinement, scarta i tweet sotto una certa *threshold*. Questo valore di *threshold* è stato cosi calcolato:

$$threshold = \theta \times \max\_value$$

dove, $\theta = [0,0.1,0.2,...,1]$ e $\max\_value$ è il valore massimo di "score" estratto dalla lista dei tweet con co-occorrenze dei termini, rispetto alle chiavi iniziali di ricerca, uguali ad 1.

Per scegliere il valore di soglia che meglio si adatta ai dati in ingresso sono stati testati più valori di soglia sullo stesso dataset andando a valutare qualitativamente i dati risultanti dall'output dell'algoritmo di *Topic Refinement*.

In fase preliminare di sperimentazione è stato scelto di svolgere gli esperimenti su tutti i dataset, descritti in precedenza, ma già elaborati dalla prima fase di post-processing.

I risultati sperimentali condotti sul primo dataset, la partita di NBA tra Huston Rockets e Miami Heat, sono riportati in Tabella 1. Si può notare come il valore di *threshold* per questo dataset vada da 0 a 225, questo significa che all'interno dei tweet sono state trovate molte parole simili e con una certa importanza. Inoltre i dati presentano una grado di chiarezza elevato anche per soglie

basse, mentre quello che influisce molto è la pertinenza dei tweet sull'argomento ricercato che si degrada via via che la soglia si abbassa.

OMISSIS

I risultati sperimentali condotti sul secondo dataset, la partita di NBA tra Toronto Raptors e Orlando Magic, sono riportati in Tabella 2. Si può notare come il valore di *threshold* per questo dataset sia relativamente basso assumendo valori tra 0 a 62, ciò significa che vi sono dati molto rumorosi. Come si può notare anche se i tweet sono scritti in maniera chiara, per via del loro valore elevato di chiarezza, non risultano essere abbastanza pertinenti con l'evento ricercato.

In particolare, per questo dataset, si può notare che per il valore di $\theta = 0.5$ cioè la metà esatta della soglia vengono catturati tutti gli eventi ma vi è ancora una forte presenza di spam. Per valori di soglia alti invece si degradano molto più rapidamente, rispetto al dataset precedente, andando ad eliminare fin troppe informazioni importanti e quindi degradando pesantemente il valore di pertinenza dei tweet sull'argomento ricercato.

OMISSIS

I risultati sperimentali condotti sul terzo dataset, la partita di NBA tra Cleveland Cavaliers e Minnesota Timberwolves, sono mostrati in Tabella 3. Questo tipo di dataset risulta particolare poiché, a differenza degli altri due dataset, per i valori di θ compreso tra 0.5 e 0.7 non si assumono i valori migliori di chiarezza e pertinenza. Questo è dettato dal fatto che nel dataset in questione lo spam e i duplicati sono scritti in maniera molto più chiara e sono molto pertinenti all'argomento ricercato. Inoltre per il valore di $\theta = 0.6$, si nota come il valore di *threshold* sia alto ma allo stesso tempo il numero di tweet iniziali non si riduce molto. Questo significa che i dati estrapolati da questo dataset sono con un'elevata percentuale inerenti alle keyword iniziali impostate per la ricerca.

Visto l'obiettivo posto in questo elaborato e quindi di non sovraccaricare l'utente finale con troppe informazioni e troppe notifiche, tra lo spam si vanno a collocare anche tutte quelle notizie che sono inerenti all'argomento ricercato ma

ritenute troppo generali. Per questo motivo l'algoritmo di *Topic Refinement* su questo tipo di dataset, con i valori di soglia intermedi, tende ad avere valori bassi di chiarezza e pertinenza. Tuttavia, per poter ridurre il numero di tweet totali risulta comunque un compromesso accettabile.

Visti i risultati appena discussi, si è scelto $\theta = 0.6$ come valore di soglia da attribuire alla fase di *Topic Refinement*. Questo valore di soglia risulta essere adeguato in tutti i dataset, andando a prendere la giusta quantità di eventi discriminandoli da una buona parte di dati rumorosi. Inoltre si riescono a prendere in quasi tutti i casi i valori migliori di chiarezza e di pertinenza.

OMISSIS

Assegnato il valore di soglia alla funzione di *Topic Refinement* verrano descritti i risultati sperimentali condotti sull'algoritmo presente in letteratura, il *TLDF* la fase di *Post-Processing* e la *Topic Refinement*.

5.3. Confronto tra le modifiche apportate e il TLDF

Di seguito verrano riportati, per tutti i dataset scelti, la valutazione dei risultati sperimentali sul sistema realizzato per mettere a confronto l'algoritmo presente in letteratura con le modifiche suggerite in questo elaborato.

- *Huston Rockets - Miami Heat*: per produrre questo dataset, le parole di ricerca immesse nel sistema sono state:

  NBA, Rockets, Heat, Huston Rockets, Miami Heat

È stata una partita molto combattuta, entrambe le squadre sono rimaste tutta la partita con il punteggio in bilico ed è durata circa 2 ore e 30 minuti e soltanto nel finale Miami Heat è riuscita a vincere il match. Per tale motivo è stata una partita molto discussa su Twitter dove gli utenti hanno prodotto un grosso flusso di informazioni.

OMISSIS

Seguono i risultati sperimentali ottenuti, applicando i criteri di confronto descritti in precedenza, mettendo in relazione il *TLDF* l'algoritmo di *Post-Processing* (PP) e l'algoritmo di *Topic Refinement* (TR).

I risultati sulla *chiarezza* e sulla *pertinenza* presentati in Figura 21 e in Tabella 4, mostrano come la *Topic Refinement* tenda ad avere risultati migliori.

Il *TLDF* produce una grande mole di dati affetti dalla presenza di una quantità elevata di rumore, quindi i tweet anche se risultano essere scritti in maniera chiara presentano un grado di pertinenza sull'argomento cercato abbastanza basso.

Aggiungendo al TLDF l'algoritmo preliminare di Post-Processing, la mole dei dati viene ridotta notevolmente ma la presenza di rumore nelle informazioni permane dando sempre come risultato un valore di pertinenza ancora basso.

Nella fase finale aggiungendo anche l'algoritmo di *Topic Refinement* si riesce ad eliminare una buona parte dei tweet che risultano non essere pertinenti con l'argomento di interesse ricercato. Infatti, si denota soprattutto come il valore di pertinenza migliori notevolmente andando a catturare più della metà dei tweet, del set in questione, pertinenti con l'argomento ricercato.

	Chiarezza	Pertinenza
TLDF	0.64	0.21
TLDF + PP	0.70	0.38
TLDF + PP + TR	0.83	0.69

Tabella 4. Confronti della *chiarezza* e della *pertinenza* sul primo dataset per il Twitter Live Detection Framework, l'algoritmo di Post-Processing e l'algoritmo di Topic Refinement.

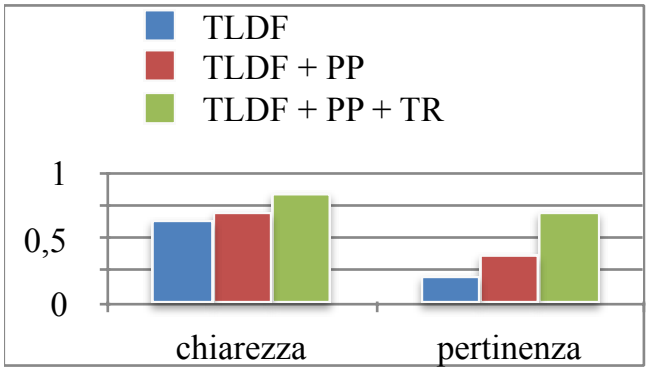


Fig. 21. Confronti della *chiarezza* e della *pertinenza* sul primo dataset per il Twitter Live Detection Framework, l'algoritmo di Post-Processing e l'algoritmo di Topic Refinement.

OMISSIS

Il numero degli eventi estratti dalla *Topic Refinement* risulta di gran lunga inferiore al *TLDF*, questo significa che alcuni degli eventi significativi non vengono captati aggiungendo la *Topic Refinement*. Tuttavia questo risultato può essere tollerato, in quanto, per il sistema presentato in questo elaborato non risulta conveniente avvisare l'utente finale con un gran numero di informazioni rilevanti ma generiche come accade con il *TLDF*. Per tale motivo, il sistema presentato in questo elaborato riesce a discriminare eventi importanti molto più dettagliati a discapito della perdita di alcuni eventi importanti.

	Evento	Spam	Duplicato
TLDF	312	1971	1660
TLDF + PP	25	357	222
TLDF + PP + TR	18	46	52

Tabella 5. Numeri di *Evento*, *Spam* e *Duplicato* sul primo dataset per il Twitter Live Detection Framework, l'algoritmo di Post-Processing e l'algoritmo di Topic Refinement.

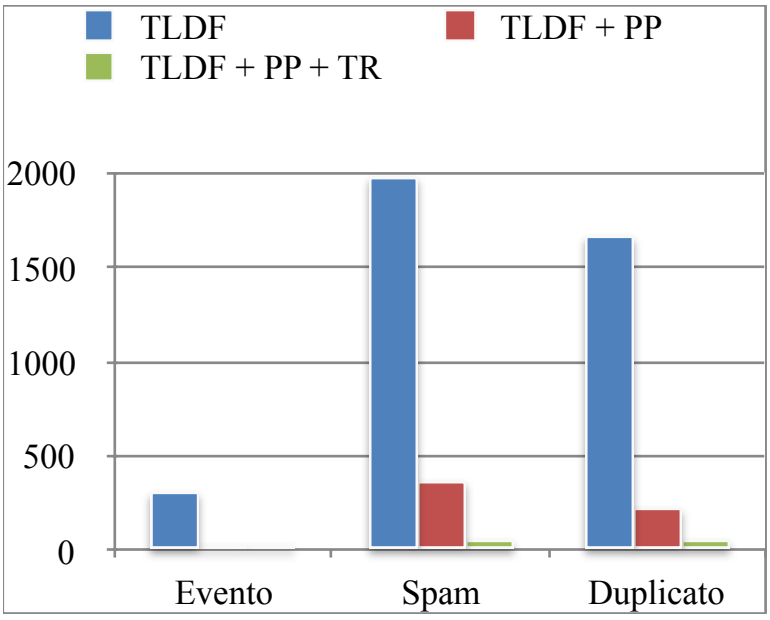


Fig. 22. Confronti tra *Evento, Spam, Duplicato* sul primo dataset per il Twitter Live Detection Framework, l'algoritmo di Post-Processing e l'algoritmo di Topic Refinement.

Nella tabella sottostante sono riportati gli eventi ottenuti applicando l'algoritmo di Topic Refinement. Si riescono a discriminare correttamente le

formazioni iniziali, alcune statistiche sui giocatori, qualche commento sulla partita e il risultato della partita sia durante i quarti che alla fine di ogni quarto.

orario	tweet
18-01-2017 01:35:31	Our Heat starting lineup tonight against Houston: Dragić , Waiters, McGruder, Babbitt, and Whiteside. Legacy jersey…
18-01-2017 01:41:03	Rockets com: G - Bev G - Harden F - Brewer F - Ariza C - Capela Heat com: Dragic, Waiters, Babbit, McGruder e Whiteside.
18-01-2017 01:42:32	Rockets open defensively with Harden on Babbitt, as if to cut him a break on that end. Heat open with McGruder on Harden.
18-01-2017 01:46:14	GUARDED: Rockets 8, Heat 6, 1st Quarter - 8:07 - RUWTbot added 6 roots
18-01-2017 02:04:11	Rockets 32, #Heat 27 after one. Harden already with 10 points, 6 assists. Waiters leads Miami with 8; Whiteside, Dragic have 6 each.
18-01-2017 02:12:22	Heat on 7-0 run to the lead as Rockets repeatedly miss at the rim. But it's like they always say, live by the layup, die by the layup.
18-01-2017 02:24:21	GUARDED: Rockets 48, Heat 46, 2nd Quarter - 4:10 - RUWTbot added 8 roots
18-01-2017 02:29:27	Whiteside, Dragic & Waiters all scoring in double figures to give Heat a 53-48 lead over Houston with 2:14 left in 1st half on @NBATV
18-01-2017 02:34:37	Rockets 53, Heat 53 at half. Harden with 15 for Rockets, Whiteside with 11 points, 8 rebounds for Heat.
18-01-2017 02:53:24	NBA: Rockets 57, Heat 56, 3rd Quarter - 10:32 - RUWTbot added 8 roots
18-01-2017 02:57:34	NBA: Heat 63, Rockets 62, 3rd Quarter - 8:57 - RUWTbot added 8 roots
18-01-2017 03:17:54	31 points, 9 boards, 8 assists for James Harden. With Anderson out and Gordon (3-13 FG) struggling, he's carrying Rockets. 79-79 through 3Q.
18-01-2017 03:26:19	#HOUvsMIA: HEAT open the 4th on a 9-4 run to force a Rockets timeout! Miami's bench outscoring Houston's 34-25.
18-01-2017 03:28:29	MIA bench has 34 pts to HOU 25 - 5 different Heat players have scored in double digits (Dragic, Whiteside, Waiters, T. Johnson & J. Johnson)
18-01-2017 03:33:00	D'Antoni livid Harden did not get a call on a drive. Kane Fitzgerald T'd him up. Rockets down 5, Heat going to the line for 3 fts.
18-01-2017 03:45:10	Harden missed 3. Ariza missed 3. Dragic scores and Ellington hits the dagger. Heat up 11, 1:41 left. Rockets 8 of 36 from deep.
18-01-2017 03:53:14	Harden just got his triple double. has 37 pts., 11 rebs, 10 assists. 11.8 to play. Heat up 109-100.
18-01-2017 03:57:27	Final: #Heat 109, Rockets 103. Dragic 21 pts, Ellington 17. Whiteside 14 pts, 15 rebs. Harden with 40 pts, 12 rebs. 10 assists.

Tabella 6. Eventi estratti dalla Topic Refinement durante la partita *Rockets - Heat.*

- *Toronto Raptors - Orlando Magic*: per produrre questo dataset, le parole di ricerca immesse nel sistema sono state:

NBA, Raptors, Magic, Toronto Raptors, Orlando Magic

Questa partita a differenza della prima, non è stata ne molto combattuta ne molto discussa su Twitter, inoltre la partita è durata circa 2 ore e 30 minuti. Tuttavia le due squadre durante i primi due quarti si sono equivalse, mentre verso la metà del terzo quarto i *Magic* hanno prevalso portandosi ad un vantaggio sostanziale per poi vincere la partita.

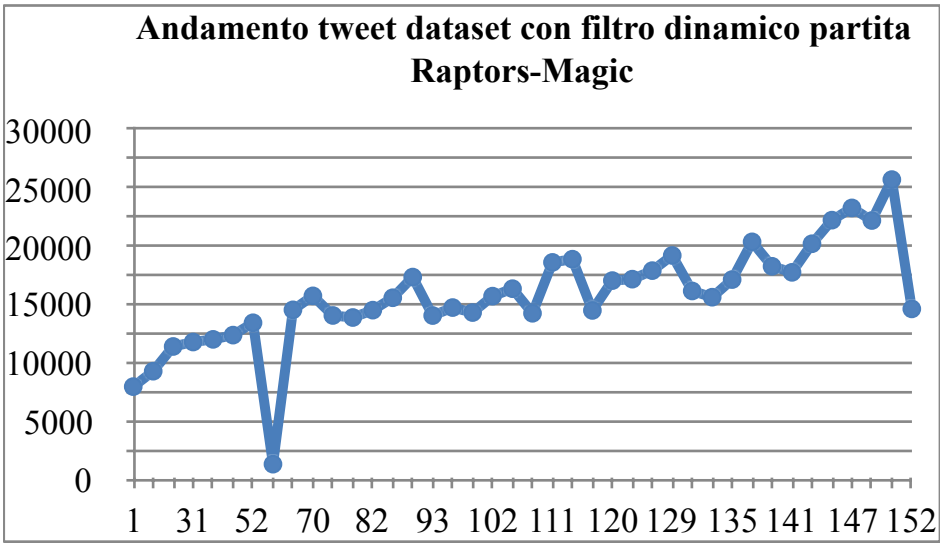


Fig. 23 Andamento dei tweet durante la partita tra *Toronto Raptors* e *Orlando Magic*. Sull'asse *x* ritroviamo i minuti trascorsi dall'inizio della partita; Sull'asse *y* i tweet acquisiti durante la partita.

Infatti, come si può vedere in Figura 23, il numero dei tweet scaricati durante l'intera partita rimane confinato tra i 10000 e i 20000, dovuto sia ad una limitazione dei servizi che Twitter impone in caso di congestione, che dalla partita la quale non ha dato molte emozioni e quindi vi è stata poca presenza di tweet in rete. Si nota che solo verso il finale di partita si hanno dei picchi verso i 25000 tweet e quando termina scende subito verso i 15000 tweet.

Una cosa molto importante che si nota su questo secondo dataset è l'uso della finestra dinamica. Nel primo dataset per tutta la durata della partita, visto che l'evento è stato molto commentato dagli utenti, la finestra dinamica rimane confinata in intervalli di 1 minuto. Mentre per questo dataset visto che gli utenti non hanno scritto molto in merito all'incontro, la finestra dinamica per le prima ora e mezza della partita rimane sempre su intervalli di tempo da 5 e 10 minuti, soltanto verso il terzo quarto e il finale di partita la finestra si riduce ad 1 minuto, infatti si nota anche l'incremento del numero dei tweet scaricati. Questo mette in risalto come, nel sistema di Live Detection presentato, l'uso della finestra dinamica

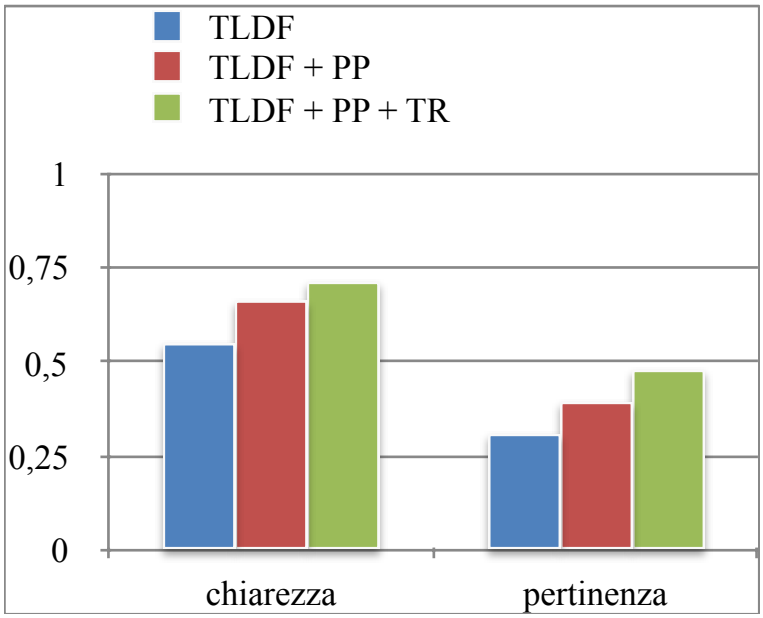


Fig. 24 Confronti della *chiarezza* e della *pertinenza* sul secondo dataset per il Twitter Live Detection Framework, l'algoritmo di Post-Processing e l'algoritmo di Topic Refinement.

permette di acquisire ed analizzare i dati in tempo reale in maniera coerente rispetto all'uso della finestra statica.

Inoltre, la natura particolare del nome di una delle due squadre "Magic", ha introdotto nei dati un valore eccessivo di rumore andando a scaricare diversi messaggi contente la parola in questione ma non pertinenti all'argomento ricercato. Questo è il motivo per il quale in questo dataset il valore di *pertinenza* rimane, per tutti e 3 i tipi di algoritmi, notevolmente basso.

Seguono i risultati sperimentali ottenuti applicando i criteri di confronto descritti in precedenza, mettendo in relazione il *TLDF* aggiungendo in un primo momento l'algoritmo di *Post-Processing* (PP) e infine aggiungendo l'algoritmo di *Topic Refinement* (TR).

Anche in questo caso, i risultati sulla chiarezza e sulla pertinenza, presentati in Figura 24 e in Tabella 7, mostrano come l'algoritmo di *Topic Refinement* tenda ad avere risultati migliori. Come descritto precedentemente notiamo che, per tutti e tre gli algoritmi, il valore di pertinenza rimane basso dovuto all'eccessiva presenza di rumore nei dati causata dalla ricerca dei termine "magic".

Sul set finale dei tweet estratti dalla *Topic Refinement* sicuramente il valore di *chiarezza* migliora e riusciamo a trovare una valore di *pertinenza* in cui almeno la metà dei tweet estratti sono pertinenti con l'argomento ricercato.

	Chiarezza	Pertinenza
TLDF	0.55	0.31
TLDF + PP	0.66	0.39
TLDF +PP + TR	0.71	0.48

Tabella 7. Confronti della *chiarezza* e della *pertinenza* sul secondo dataset per il Twitter Live Detection Framework, l'algoritmo di Post-Processing e l'algoritmo di Topic Refinement.

Mentre dai confronti tra Evento, Spam e Duplicato, mostrati in Figura 25 e Tabella 8, si evince che il *TLDF* presenta una gran mole di dati su cui vi è un'alta presenza di rumore. Inoltre, vengono identificati un alto numero di eventi rispetto a quelli estratti aggiungendo la fase di *Post-Processing* e la fase di *Topic Refinement*. Questo numero elevato di eventi però va accostato anche ad un alto numero di duplicati ed un'alto numero di spam. Nel sistema presentato in questo elaborato, per poter estrarre eventi importanti al fine di informare l'utente, non si può accettare un valore così elevato. Quindi, a discapito della perdita di eventi importanti ma non ai fini di dover notificare l'utente, si accetta la perdita di queste informazioni. Risulta adeguato quindi il valore di eventi assegnati aggiungendo l'algoritmo di Topic Refinement, poiché riduce notevolmente la presenza dei duplicati e dello spam.

	Evento	Spam	Duplicato
TLDF	228	2363	1514
TLDF + PP	16	281	116
TLDF + PP + TR	15	104	29

Tabella 8. Numeri di Evento, Spam e Duplicato sul secondo dataset per il Twitter Live Detection Framework, l'algoritmo di Post-Processing e l'algoritmo di Topic Refinement.

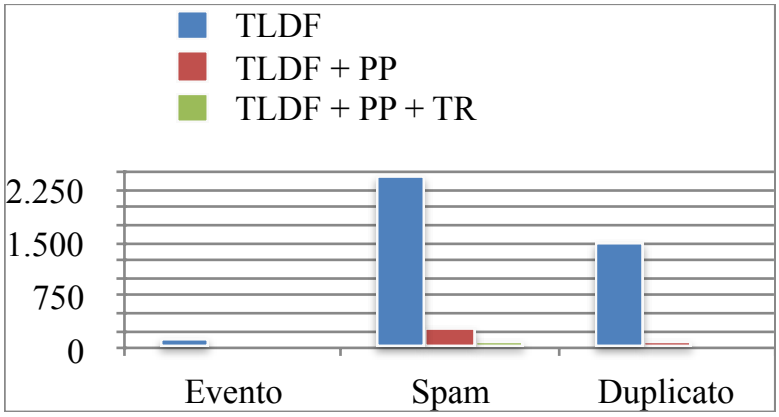


Fig. 25. Confronti tra *Evento, Spam, Duplicato* sul secondo dataset per il Twitter Live Detection Framework, l'algoritmo di Post-Processing e l'aalgoritmo di Topic Refinement.

In Tabella 9 sono riportati gli eventi ottenuti aggiungendo al *TLDF* sia l'algoritmo di *Post-Processing* che l'algoritmo di *Topic Refinement*.

Come si può notare, nella fase precedente l'inizio della partita si riesce a captare la formazione iniziale di una delle due squadre e l'assenza di un giocatore importante perché infortunato, la partita era programmata per le ore 01:00:00 (orario italiano). Dall'inizio della partita passano circa 43 minuti prima che si riesca ad avere un'altro evento, questo è dovuto al fatto che la partita non è stata seguita da molti utenti, quindi vi era poca presenza di informazioni. Mentre nel finale, precisamente durante gli ultimi due quarti, in cui le due squadre stanno per concludere il match, si vede che la presenza di eventi aumenta. Molti di questi tweet riportano il fatto che una delle due squadre stia giocando veramente male e infine si riesce anche a captare il risultato finale.

La particolarità di questo dataset è dovuta anche alla presenza di informazioni riguardo la partita ma catturando la prospettiva dell'utente che scrive. Ciò mette in luce la particolarità del *TLDF* e dal sistema realizzato in questo elaborato che cerca anche di catturare gli eventi dalla prospettiva dell'utente.

orario	tweet
04-02-2017 00:29:09	ProBallMetrics: Lineup note: Raptors will start Lowry, Powell, Carroll, Patterson, Valanciunas on Friday. #DraftKings #FanDuel #fantasy #NB…
04-02-2017 00:38:15	JUST IN: Raptors announce that DeMar DeRozan's ankle injury will keep him out vs. Magic tonight. DeRozan missing 6t…
04-02-2017 01:43:25	Magic lead the Raptors 28-24 at the end of the 1st. TOR: J Valanciunas 9pts, 1reb, 0ast ORL: S Ibaka 12pts, 4reb, 0ast (ESPN) …
04-02-2017 01:47:09	RT OrlandoMagic: End 1Q: Magic 28, Raptors 24 👏 📺: FOXSportsFL 📻: ESPNOrlando #LetsGoMagic
04-02-2017 01:52:08	GUARDED: Raptors 36, Magic 34, 2nd Quarter - 7:24 - RUWTbot added 8 roots
04-02-2017 02:08:09	Raptors lead the Magic 54-52 at halftime. TOR: J Valanciunas 13pts, 3reb, 1ast ORL: S Ibaka 14pts, 6reb, 1ast (ESPN) …
04-02-2017 02:37:01	So basically without Demar Derozan the Raptors are trash.

orario	tweet
04-02-2017 02:43:12	SmithRaps: Raptors have 5 points in 9 1/2 minutes, Vucevic has 14. Discuss
04-02-2017 02:51:13	Magic lead the Raptors 73-66 at the end of the 3rd. TOR: J Valanciunas 14pts, 7reb, 1ast ORL: S Ibaka 18pts, 9reb, 1ast (ESPN) …
04-02-2017 03:02:36	The Raptors are playing so poorly, Fred VanVlett is their leading scorer in the 4th.
04-02-2017 03:11:20	NBA: Magic 89, Raptors 84, 4th Quarter - 4:34 - RUWTbot added 5 roots
04-02-2017 03:12:14	NBA: Magic 91, Raptors 84, 4th Quarter - 3:57 - RUWTbot took away 5 roots (Close Finish) https://t.co/yi4Hh7CQBR
04-02-2017 03:34:22	Serge comes up big for the OrlandoMagic. He pours in 20p, 12r, & 3b for ORL in their 102-94 W over the Raptors
04-02-2017 03:34:46	Raptors and Dwane Casey are going through a very rough stretch. 2-8 in their L10 games😵. Imagine if DeRozan didn't resign 😯
04-02-2017 03:37:23	Final: Magic 102 Raptors 94. TOR: J Valanciunas 18pts, 11reb, 1ast ORL: S Ibaka 20pts, 12reb, 1ast (ESPN)

Tabella 9. Eventi estratti dall'algoritmo di Topic Refinement durante la partita Raptors - Magic.

- *Cleveland Cavaliers - Minnesota Timberwolves*: per produrre questo dataset le parole di ricerca immesse nel sistema sono state:

  NBA, Cavaliers, Timberwolves, Cleveland Cavaliers, Minnesota Timberwolves

Si è scelto di acquisire i dati di una partita dei Calvaliers perché oltre ad essere nelle prime posizioni della *Estern Conference* (nel periodo durante lo sviluppo dei dati sperimentali), sono gli attuali detentori del titolo *NBA* e in squadra sono presenti giocatori del calibro di LeBron James. Tuttavia, anche se la partita è stata molto combattuta e ha visto soltanto nel finale prevalere i Cavaliers non è stata molto discussa su Twitter.

OMISSIS

Si è scelto di mostrare questo dataset per la sua natura particolare rispetto ai precedenti. Infatti, questo dataset non solo presenta un numero inferiore di tweet

rispetto agli altri precedentemente mostrati ma si riescono a catturare lo stesso numero o un numero maggiore di eventi. Ciò mostra come in una partita meno discussa, quindi con poche informazioni, si riescono ad estrapolare in ogni caso argomenti di interesse. Questo è dovuto, sia alla presenza della finestra dinamica, sia al tipo di informazioni che gli utenti condividono. Infatti, specialmente su questo dataset, si è notato che molti tweet categorizzati come *spam* si riferissero comunque ai giocatori di una delle due squadre o ad un'azione saliente della partita commentata dagli utenti. Per il sistema presentato in questo elaborato, in cui il numero di eventi da notificare all'utente finale si deve ridurre, molte di queste informazioni sono state considerate *spam* e quindi hanno fatto diminuire il valore di pertinenza.

Seguono i risultati sperimentali ottenuti applicando i criteri di confronto descritti in precedenza e mettendo in relazione il *TLDF* aggiungendo in un primo momento l'algoritmo di *Post-Processing* (PP) e infine aggiungendo l'algoritmo di *Topic Refinement* (TR).

Come si può notare in Tabella 10 e in Figura 27, i valori migliori di *chiarezza* e *pertinenza* si ottengono aggiungendo al *TLDF* soltanto l'algoritmo di Post Processing. Ciò significa che più della metà dei tweet appartenenti al dataset è pertinente all'argomento cercato e i dati risultano essere scritti in maniera chiara. Andando ad applicare l'algoritmo di *Topic Refinement* viene leggermente degradato il valore di *pertinenza*, poiché per ridurre la quantità dei dati vengono eliminate informazioni pertinenti con l'argomento ricercato, ma non così interessanti da categorizzarli come evento e notificare l'utente finale. Questo è un risultato atteso per via della natura dei dati acquisiti, più la finestra dinamica si adatta al flusso di informazioni più acquisisce eventi pertinenti con il set di keyword iniziali, quindi la scrematura finale porterà spesso ad un degrado della pertinenza.

	Chiarezza	Pertinenza
TLDF	0.68	0.51
TLDF + PP	0.73	0.52
TLDF + PP + TR	0.72	0.50

Tabella 10. Confronti della *chiarezza* e della *pertinenza* sul terzo dataset per il Twitter Live Detection Framework, l'algoritmo di Post-Processing e l'algoritmo di Topic Refinement.

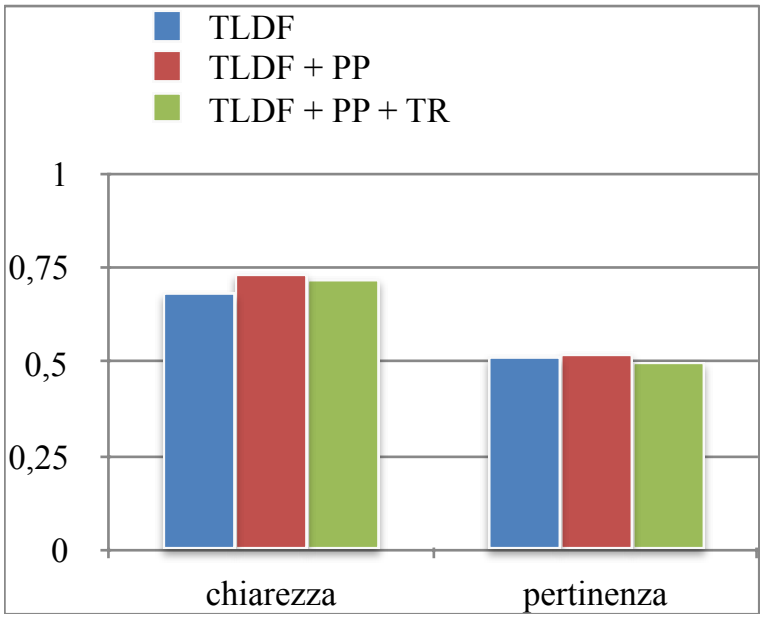


Fig. 27. Confronti della *chiarezza* e della *pertinenza* sul terzo dataset per il Twitter Live Detection Framework, l'algoritmo di Post-Processing e l'algoritmo Topic Refinement.

Mentre dai confronti tra Evento, Spam e Duplicato, mostrati in Figura 28 e Tabella 11, si evince che il *TLDF,* come nei precedenti dataset, presenta una gran mole di dati su cui vi è un'alta presenza di rumore. Inoltre vengono identificati un alto numero di eventi rispetto a quelli estratti aggiungendo la fase di *Post-Processing* e la fase di *Topic Refinement*. Su questo dataset le modifiche suggerite in questo elaborato presentano ancora una grossa mole di dati. Infatti, aggiungendo la fase di Topic Refinement, lo *spam* e i *duplicati* assumono ancora dimensioni troppo elevate per poter discriminare correttamente gli eventi. Questa particolarità è dovuta dalla presenza di numerosi termini di ricerca utilizzati durante la fase di acquisizione sui tweet analizzati. Infatti, come accennato in precedenza, se i tweet presentano più di 2 termini uguali alle chiavi di ricerca o si presentano spesso con la stessa terminologia avranno assegnato un valore alto di *score*. Assegnando un valore di soglia intermedio all'algoritmo di Topic Refinement per eliminare alcuni dei tweet estratti in questo dataset non risulta sufficiente per poter ridurre la grande quantità dei dati acquisiti. Assegnare un valore di soglia più alto potrebbe comunque portare all'eliminazione di argomenti interessanti.

	Evento	Spam	Duplicato
TLDF	301	1723	1283
TLDF + PP	22	189	133
TLDF + PP + TR	16	129	81

Tabella 11. Numero di Evento, Spam e Duplicato sul terzo dataset per il Twitter Live Detection Framework, l'algoritmo di Post-Processing e l'algoritmo di Topic Refinement.

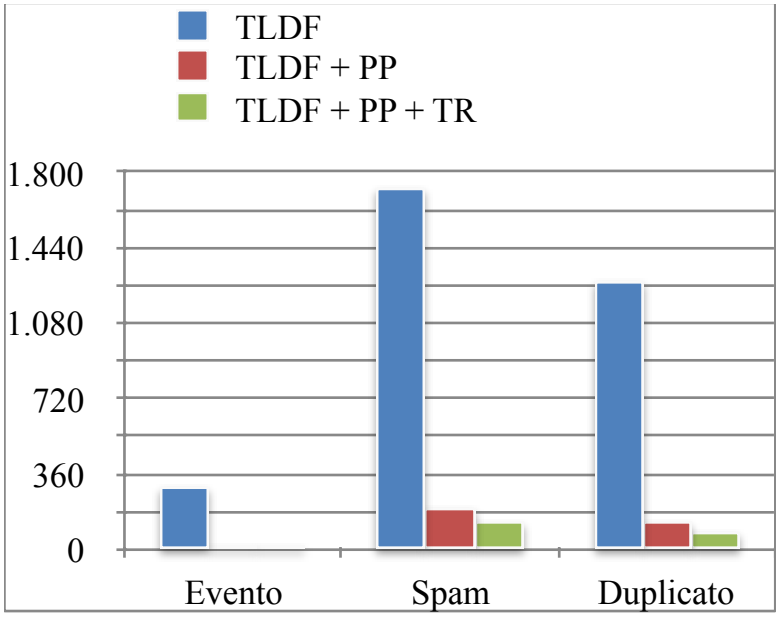


Fig. 28. Confronti tra *Evento, Spam, Duplicato* sul terzo dataset per il Twitter Live Detection Framework, l'algoritmo di Post-Processing e l'algoritmo di Topic Refinement.

In Tabella 11 sono riportati gli eventi ottenuti aggiungendo al *TLDF* sia l'algoritmo di *Post-Processing* che l'algoritmo di *Topic Refinement*.

Il primo evento che si riesce a catturare è circa 20 minuti dall'inizio della partita. Ad ogni modo, anche se la partita non è stata molto discussa si è riusciti nella parte centrale del match a discriminare diversi eventi a pochi minuti di distanza con i risultati parziali delle due squadre. Infine, si riescono anche ad evidenziare le statistiche dell'intera partita giocata da LeBron James.

orario	tweet
15-02-2017 02:20:18	GUARDED: Timberwolves 15, Cavaliers 9, 1st Quarter - 7:07 - RUWTbot added 7 roots (Individual Performance)

orario	tweet
15-02-2017 02:34:22	KAT starts the game perfectly: 6-6 from the field for a game-high 12 PTS. Timberwolves lead cavs 30-26 at the e…
15-02-2017 02:50:19	NBA: Cavaliers 44, Timberwolves 40, 2nd Quarter - 6:22 - RUWTbot took away 6 roots (Individual Performance)
15-02-2017 02:57:32	GUARDED: Cavaliers 54, Timberwolves 47, 2nd Quarter - 4:01 - RUWTbot took away 10 roots (Individual Performance)
15-02-2017 03:06:01	HALFTIME: Cavaliers 69, Wolves 61. Wiggins with a game-high 18 points. LeBrone with 14 points and 8 assists.
15-02-2017 03:33:01	ELEVATED: Cavaliers 82, Timberwolves 69, 3rd Quarter - 7:49 - RUWTbot added 26 roots (Individual Performance)
15-02-2017 03:35:23	NBA: Cavaliers 87, Timberwolves 73, 3rd Quarter - 6:24 - RUWTbot added 16 roots (Individual Performance)
15-02-2017 03:37:52	NBA: Cavaliers 88, Timberwolves 81, 3rd Quarter - 4:39 - RUWTbot took away 5 roots (Individual Performance)
15-02-2017 03:40:52	NBA: Cavaliers 88, Timberwolves 83, 3rd Quarter - 3:21 - RUWTbot added 10 roots (Individual Performance)
15-02-2017 03:45:16	GUARDED: Cavaliers 91, Timberwolves 88, 3rd Quarter - 1:17 - RUWTbot took away 20 roots
15-02-2017 03:46:25	GUARDED: Cavaliers 93, Timberwolves 88, 3rd Quarter - 0:38 - RUWTbot added 29 roots (Individual Performance)
15-02-2017 03:46:54	NBA: Cavaliers 93, Timberwolves 91, 3rd Quarter - 0:14 - RUWTbot added 12 roots (Individual Performance)
15-02-2017 03:50:30	Cavaliers are tied with the Timberwolves 93-93 at the end of the 3rd. CLE: L James 20pts, 6reb, 13ast MIN: A Wiggins 37pts, 3reb, 2ast (ESP…
15-02-2017 04:16:24	King James 👑 NAILS the clutch step back 👌 @cavs up 114-106 w/ 1:39 to go on NBA TV! #FanNight
15-02-2017 04:21:44	Final: Timberwolves 108, Cavs 116. Minnesota will be in Denver to take on the Nuggets tomorrow night at 8 p.m. CT on @fsnorth/ @wccoradio.
15-02-2017 04:23:13	NBA: LeBron (25 PTS, 14 REBS, 8 ASTS) & Kyrie (25 PTS, 7 ASTS) come up huge for the cavs & they defeat the…

Tabella 12. Eventi estratti dall'algoritmo di *Topic Refinement* durante la partita *Cavaliers - Timberwolves*.

6. Conclusioni

I risultati sperimentali prodotti hanno mostrato come l'implementazione di un sistema automatizzato per la ricerca, l'analisi e la valutazione dei dati estratti da Twitter al fine di informare l'utente finale su un evento specifico ha prodotto ottimi risultati. Appurata la superiorità del *Twitter Live Detection Framework* per ricercare eventi in tempo reale attraverso l'uso di una finestra di acquisizione dinamica con l'aggiornamento costante delle parole di ricerca.

Si è proceduto con l'elaborazione di una prima fase di *Acquisizione* per poter ricercare e scaricare i dati su Twitter. Successivamente si è passati ad una seconda fase di analisi per poter ridurre il più possibile la presenza di rumore nelle informazioni acquisite e quindi estrarre argomenti di interesse agglomerandoli in Topic. Questa fase di sviluppo ha portato alla realizzazione di un'interfaccia web che permettesse di rendere il sistema fruibile a chiunque e automatizzarlo, collegando inoltre le due fasi di acquisizione e di analisi in un unico sistema di *Live Detection.* In questa parte di sistema elaborato, non solo si acquisiscono gli argomenti di tendenza ma si analizzano, si aggregano in Topic e si aggiornano le chiavi di ricerca per poi riavviare la ricerca. Inoltre, durante la fase di sperimentazione, si è realizzato un'ulteriore sistema di scrematura dei dati estratti dalla fase di analisi in tempo reale, coiè la fase di *Post-Processing* e la fase di *Topic Refinement*.

In questa fase si è sviluppato l'algoritmo di *Post-Processing* e l'algoritmo di *Topic-Refinement*, per andare a catturare il più possibile eventi mirati al fine di notificare l'utente scartando la presenza di informazioni ridondanti e di spam. Si è quindi verificata la loro efficacia applicandoli in aggiunta al *TLDF* comparando i risultati di tutte e tre i sistemi realizzati a parità di configurazione (stesso dataset). Si sono utilizzate le metriche, introdotte nel paragrafo 5.1, al fine di evidenziare gli aspetti ritenuti più importanti per il problema in esame. Tali dati hanno messo in evidenza la superiorità delle modifiche suggerite, che si sono dimostrate molto più mirate per il problema in esame andando ad eliminare una grossa mole di dati, specialmente eventi duplicati e spam.

Infine, il sistema finale realizzato, le cui modifiche rispetto allo stato dell'arte sono state prodotte nel capitolo 3, si è rilevato essere il migliore di tutti per l'obiettivo finale della tesi. Questo risultato è dovuto non solo alla scelta

dell'utilizzo del *TLDF*, ma anche ai due algoritmi di Post-Processing e Topic Refinement proposti che ne hanno esteso le capacità rendendolo più performante per il problema in esame. Inoltre l'interfaccia web che è stata realizzata permette all'utente di accedere a tutte le fasi del sistema in maniera chiara e semplice.

In conclusione, questo progetto ha raggiunto l'obiettivo che si era prefissato: non solo si è utilizzata una finestra dinamica di ricerca che si adatta correttamente agli eventi in tempo reale, ma inoltre si è ridotto enormemente il numero dei dati estratti durante la fase di analisi andando a trovare eventi inerenti all'argomento di interesse ricercato. Certamente, si può tentare di ridurre ulteriormente la presenza di rumore all'interno dei dati e contemporaneamente aumentare la presenza di eventi che vengono persi durante la fase di *Post-Processing* e di *Topic Refinement*.

Viste le limitazioni evidenziate, specialmente sul terzo dataset, dall'algoritmo di Topic Refinement come possibili evoluzioni future si può procedere in due modi differenti.

Il primo modo sicuramente è migliorare l'algoritmo di Topic Refinement, andando ad analizzare i termini presenti nei Topic scartati dall'algoritmo. Non solo per arricchire il risultato finale aggiungendo argomenti di interesse, ma per discriminare ulteriormente la presenza di spam e duplicati.

Il secondo modo, viste le limitazioni sulla scelta della soglia dell'algoritmo di Topic Refinement in base al tipo di dati acquisiti. Essa risulta essere limitante e non eccessivamente accurata nel discriminare i Topic. Quindi, si potrebbe evitare di utilizzare una soglia per eliminare i Topic meno rilevanti ma cercare di aggregarli il più possibile. Aggregarli in maniera da ottenere un numero inferiore di Topic ma con un elevato numero di eventi rispetto allo spam e ai duplicati. Questo metodo di aggregazione potrebbe, per esempio, raggruppare tutti quei Topic la cui struttura si presenta nella stessa forma.

Indice delle Figure

Indice delle Tabelle

Riferimenti Bibliografici:

[1] Luca Maria Aiello, Giorgio Petkos, Carlos Martin, David Corney, Symeon Papadopoulos, Ryan Skraba, Ayse Goker, Yiannis Kompatsiaris, Alejandro Jaimes, *Sensing trending topics in Twitter, Multimedia*, IEEE Transaction on 15 (6) (2013)

[2] H. Wu, R. Luk, K. Wong w K. Kwok, *Interpreting TF-IDF term weights as making relevance decisions*

[3] J. Parker, Y. Wei, A. Yates, O. Frieder, N. Goharian, A *framework for detecting public health trends with Twitter*

[4] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, Edward Chang, *PFP: Parallel FP-Growth for Query Racommendation*

[5] Georgios Petkos, Symeon Papadopoulos, Luca Maria Aiello, Ryan Skraba, Yiannis Kompatsiaris, *A soft frequent pattern mining approach for textual topic detection*

[6] David M. Blei, Andrew Y. Ng e Michael I. Jordan, *Latent Dirichlet Allocation*

[7] S. Phuvipadawat e T.Murata, *Breaking news detection and tracking in Twitter*

[8] B. O'Connor, M. Krieger, D. Ahn, *TweetMotif: Exploratory search and topic summarization for Twitter*

[9] H. Becker, M. Naaman, L. Gravano, *Beyond trending topics:Real-world event identification on Twitter*

[10] J. Sankaranarayanan, H. Samet, B. E. Teitler, M. D. Lieberman, J. Sperling, *Twitter stand: News in tweets*

[11] G. P. C. Fung, J. X. Yu, P. S. Yu e H. Lu, *Parameter free bursty events detection in text streams*

[12] James Allan, Victor Lavrenko, Daniella Malin e Russell Swan, *Detections, Bounds, and Timelines: UMass and TDT-3*

[13] S. Petrović, M. Osborne, e V. Lavrenko, *Streaming first story detection with application to Twitter*, Stroudsburg, PA, USA, 2010

[14] J. Han, H. Pei, and Y. Yin. *Mining Frequent Patterns without Candidate Generation*

[15] X.Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger, *SCAN: A structural clustering algorithmfor networks*

[16] B. Goethals, *Frequent Set Mining*

[17] Salvatore Gaglio, Giuseppe Lo Re, Marco Morana, *A framework for real-time Twitter data analysis*

[18] Salvatore Gaglio, Giuseppe Lo Re, Marco Morana, *Real-Time Detection of Twitter Social Events from the User's Perspective*

[19] http://www-nlp.stanford.edu/software/CRF-NER.shtml

[20] J. Han, C. Moraga, *The Influence of the sigmoid function parameters on the speed of back propagation learning*

[21] *https://dev.twitter.com/*

[22] *https://dev.twitter.com/docs/api/streaming*

[23] *https://apps.twitter.com/*

[24] *https://dev.twitter.com/streaming/overview/connecting*

[25] *http://twitter4j.org/en/index.html*

[26] *http://web.resource.org/rss/1.0/*

[27] *http://getbootstrap.com/*

[28] *http://www.chartjs.org/*

[29] Alessandra De Paola, Pierluca Ferraro, Salvatore Gaglio, Giuseppe Lo Re, Sajal K. Das, *An Adaptive Bayesian System for Context-Aware Data Fusion in Smart Environments*. In IEEE Transactions on Mobile Computing.

[30] Giuseppe Lo Re, Fabrizio Milazzo, and Marco Ortolani, *A distributed Bayesian approach to fault detection in sensor networks*. Global Communications Conference (GLOBECOM), 2012 IEEE. IEEE, 2012.