UNIVERSITÀ
DEGLI STUDI
DI PALERMO

Progettazione e Sviluppo di un Sistema per il Riconoscimento di Account Malevoli sui Social Network

Tesi di Laurea Magistrale in Ingegneria Informatica

C. Ruocco

Relatore: Prof. Giuseppe Lo Re
Correlatori: Ing. M. Morana

Progettazione e Sviluppo di un Sistema per il Riconoscimento di Account Malevoli sui Social Network

Tesi di Laurea di:

Dott. Claudio Ruocco

Relatore:

Ch.mo Prof. Giuseppe Lo Re

Controrelatore:

Ing. Rosario Sorbello

Correlatore:

Ing. Marco Morana

Sommario

I *social network*, come Twitter e Facebook, hanno conosciuto un incremento enorme della loro popolarità negli ultimi anni. La facilità di accesso ai *social* attraverso qualsiasi tipo di *smartphone*, tramite *app* dedicate, ha contribuito in maniera sostanziale alla loro diffusione, rendendoli sempre disponibili in modo semplice e veloce, in qualunque posizione l'utente si trovi.

Questa popolarità ha, ovviamente, attirato l'interesse di *cyber-criminali*: il fenomeno dello *spam* sui *social network* è diventato sempre più di rilievo. Nuovi tipi di *spammer* sono infatti emersi, i *social spammer*, utenti che si fingono genuini e attirano quanti più followers possibili, per poi avere un *target* di utenti di dimensione considerevole verso cui indirizzare azioni di disturbo.

Diversi sono stati i casi di attacchi sui *social network* che hanno attirato l'attenzione dei media, non ultimo quello degli account falsi, *fake accounts*, utilizzati per influenzare l'opinione pubblica durante le elezioni presidenziali negli Stati Uniti del 2016.

Gli stessi gestori dei *social network* stanno prevedendo misure sempre più stringenti per contrastare tale fenomeno. Twitter, ad esempio, ha reso necessaria l'autenticazione di ogni *account* mediante *email* o numero di cellulare e sta sviluppando un algoritmo di *machine learning* in grado di identificare automaticamente *account* potenzialmente malevoli.

In quest'elaborato viene presentato un sistema di identificazione automatica che cerca di sfruttare caratteristiche peculiari degli *spammer* per individuarli tra migliaia di utenti dal comportamento corretto, detti genuini. Considerata, inoltre, la difficoltà di avere a disposizione dati annotati su cui addestrare e testare sistemi automatici di *spam* detection, si propone inoltre un sistema semi-automatico di etichettatura mediante tecniche di *clustering*.

INDICE

INTRODUZIONE **1**
CAPITOLO 1: Lo Stato dell'Arte **3**
CAPITOLO 2: I Social Network **7**
2.1 – Twitter 7
2.1.1 - Panoramica API 9
2.1.2 - Condivisione Dati e Privacy 12
2.1.3 - Spam su Twitter 13
CAPITOLO 3: Il Sistema Semiautomatico di Annotazione **15**
3.1 - Costruzione del Dataset 15
3.1.1 - Ricerca 17
3.1.2 - Espansione 18
3.2 - Algoritmi per l'annotazione del dataset 19
3.2.1 - Analisi degli URL 19
3.2.2 - Near Duplicates Clustering 19
3.2.3 - Quality Threshold Clustering 27
CAPITOLO 4: Risultati Sperimentali **32**
4.7.1 - Random Forest 36
4.7.2 – C4.5 37
4.7.3 – Implementazione 39
CAPITOLO 5: Conclusioni e Lavori Futuri **40**
INDICE DELLE FIGURE **42**
INDICE DELLE TABELLE **43**
BIBLIOGRAFIA **44**
SITOGRAFIA **46**

INTRODUZIONE

I *social network*, come Twitter e Facebook, hanno conosciuto un incremento enorme della loro popolarità negli ultimi anni. Secondo recenti studi, sono diventati la principale fonte di informazione per circa il 30% degli americani [1]. Inoltre, la grande diffusione di *smartphone,* oltre a favorire la nascita di nuovi paradigmi di *sensing* [2][3], ha aumentato la facilità di accesso ai *social*, rendendoli sempre disponibili in modo semplice e veloce, in qualunque posizione l'utente si trovi.

Questa popolarità ha, ovviamente, attirato l'interesse di *cyber-criminali*: Bilge et Al. [4] hanno dimostrato la semplicità con cui è possibile, per i *cyber-criminali*, lanciare un attacco automatizzato per rubare informazioni dettagliate su chiunque. Inoltre, diversi attacchi sono stati portati avanti con motivazioni politiche: nel 2016, in concomitanza con le elezioni presidenziali degli Stati Uniti, è stato scoperto l'utilizzo di account falsi, *fake accounts*, col solo scopo di manipolare l'opinione pubblica e, di conseguenza, i risultati elettorali [24].

Inoltre, una nuova categoria di *cyber-criminali* è emersa nei *social network*, ovvero i *social spammer*, utenti il cui unico obiettivo è quello di raccogliere quanti più *followers* possibili, utenti iscritti per ricevere tutti gli aggiornamenti di stato pubblicati dallo *spammer* in questione, in modo da avere un *target* di utenti di dimensione considerevole verso cui indirizzare azioni di disturbo. Spesso, inoltre, questi *spammer* si fingono *hub* di informazioni, impersonando fonti di informazione fidate, e condividono *link* malevoli solo di tanto in tanto.

D'altra parte, sebbene negli ultimi mesi gli stessi gestori dei *social network* stiano prendendo contromisure sempre più stringenti per combattere i *cyber-criminali* (nel solo mese di giugno 2018 Twitter ha acquisito una *start-up* di sicurezza informatica, Smyte, ed ha reso obbligatoria l'autenticazione via *email* o cellulare), risulta evidente la necessità di metodi automatici in grado di rilevare utenti malevoli nel minor tempo possibile.

La stessa Twitter sta sviluppando un algoritmo di *machine learning* in grado di fare ciò: secondo dati recenti, questo algoritmo ha identificato una media di quasi 10 milioni di casi di potenziali *spammer* a settimana nel solo mese di maggio 2018. Altri metodi, presenti in letteratura, sono brevemente introdotti nel capitolo 1, Lo Stato dell'Arte.

Tuttavia, un qualunque metodo di identificazione automatizzata ha bisogno di un *dataset* già etichettato che funga da *ground-truth*, ovvero da base con cui confrontare i propri risultati per effettuare le necessarie fasi di addestramento e *testing*. Ma, data la veloce evoluzione dei *social network* e le stringenti *policy* sulla *privacy* di Twitter, in rete sono pochi i *dataset* presenti, etichettati e con la maggior parte dei dati ancora reperibili.

Per questo motivo, prima di proporre un metodo di identificazione di utenti malevoli, *spammer*, è stato necessario costruire un nuovo *dataset*, utilizzando le API di Twitter ed euristiche di ricerca di dati in grado di restituire quanti più utenti *spammer* possibili. Il dataset finale utilizzato è costituito da circa 8 milioni di *tweet*, divisi fra 40.000 utenti.

Questo elaborato si propone, quindi, l'obiettivo di presentare una nuova metodologia di identificazione di *spammer*, che sia quanto più automatizzata possibile e che riduca lo sforzo di etichettatura manuale. Alla base del sistema proposto vi sono caratteristiche peculiari identificate per gli *spammer*, fra cui la tendenza a condividere URL malevoli [5], la presenza di *pattern* nella scrittura dei *tweet*, la presenza di caratteristiche comuni a più utenti. Alla luce di queste caratteristiche, il sistema proposto si articola in 3 fasi successive. Ad una prima fase di analisi degli URL seguono una fase di *Near Duplicates Clustering* ed una di *Quality Threshold Clustering*. L'analisi effettuata è spiegata nel dettaglio nel capitolo 3.

Il resto dell'elaborato è organizzato come segue: nel capitolo 2 vi è una breve introduzione dei *social network*, con particolare interesse verso Twitter, le sue API, le *policy* sulla *privacy* che applica e il problema dello *spam* su Twitter; nel capitolo 4 vengono riportati i risultati sperimentali ottenuti, vengono spiegate le calibrazioni effettuate e le metriche utilizzate. Infine, nel capitolo 5 sono presenti alcune conclusioni sul lavoro effettuato.

CAPITOLO 1: Lo Stato dell'Arte

La presenza di *spam* sul *web* è un problema risalente a metà degli anni '90, ben prima della nascita dei *social network*. Già allora, infatti, nel momento in cui divenne possibile utilizzare *internet* per scopi commerciali, cominciarono a diffondersi *email* di *spam*, ben presto divenute inevitabili e ripetitive.

Ad oggi, lo spam è stato osservato in diverse applicazioni, dalle *email* ai motori di ricerca, dai *blog* ai *social network*. Sui *social network* si è assistito anche all'emersione di un nuovo tipo di *spam*, il *social spam*, portato avanti da utenti il cui unico obiettivo è quello di avere quanti più *followers* possibili, ovvero utenti iscritti per ricevere aggiornamenti di stati e dunque target verso cui indirizzare azioni di disturbo. Di conseguenza, un certo numero di strategie per prevenire e combattere lo *spam* sono già state proposte, spesso basate su algoritmi di *machine learning*.

Uno studio sugli algoritmi di *machine learning* utilizzati nell'ambito dello *spam detection* è stato portato avanti da Chen et Al. [6]. Gli autori hanno, nell'ordine: costruito un *dataset* di circa 600 milioni di *tweet;* etichettato circa 6 milioni di *tweet* come *spam*, mediante il servizio di *Web Reputation* di Trend Micro; estratto, per ogni *tweet*, un insieme di *features* divise fra *user-based* e *tweet-based*, capaci di differenziare *tweet spam* e genuini; applicato sei dei più comuni algoritmi di *machine learning* utilizzati in letteratura (Random Forest, C4.5, kNN, Reti Bayesiane, Naive Bayes, SVM) e testato le loro performance in diversi casi, al variare di parametri come il rapporto fra *tweet* di *spam* e tweet genuini. In definitiva, i loro studi evidenziano come le capacità di classificazione di qualunque algoritmo di *machine learning* siano migliori in casi simulati piuttosto che reali, dato il grande sbilanciamento di dati fra *tweet* genuini e *tweet* di *spam*, che introduce necessariamente del *bias*.

Benevenuto et Al. [7] approcciano il problema del rilevamento degli *spammer* in sistemi di condivisione video. Utilizzando una collezione di utenti classificati manualmente, applicano un approccio di *machine learning* gerarchico, basato su SVM, per differenziare utenti genuini da utenti *spammer*. Tuttavia, come gli stessi autori evidenziano, la complessità computazionale del sistema di etichettatura è abbastanza elevata, per cui lavori futuri dovrebbero puntare a ridurla, per esempio mediante approcci semi-supervisionati.

Un altro approccio è quello di prevenire lo *spam* mediante l'utilizzo di *white-lists*, in cui ogni utente deve specificare una lista di utenti da cui desidera ricevere contenuti. Ad esempio, Garriss et Al. [7] propongono "RE - Reliable Email", un sistema di *white-listing* per *email* basato sui "*social link*", che permette di ricevere *emai*l da amici o amici-di-amici e, quindi, *bypassare* i filtri standard di *spam*.

In Sedhai et Al. [9], invece, viene proposto un sistema di annotazione di un *dataset* organizzato in quattro *step* successivi ed in grado di distinguere *tweet spam* da *tweet* genuini, detti *ham*. Gli *step* sono i seguenti:

1. **Selezione su base euristica di tweet**, ovvero scelta di *keywords* con cui accedere allo *stream* dei tweet pubblici di Twitter e con cui è più semplice trovare *spam*. Dunque, gli autori individuano un insieme di *keyword*, dette "*spammy*", legate ad argomenti come guadagno semplice di denaro e contenuti per adulti ed utilizzano queste per effettuare la costruzione del *dataset*.

2. **Annotazione manuale basata sui cluster**, ovvero utilizzo del MinHash come algoritmo di *clustering* per raggruppare contenuti simili fra loro e poter esaminare manualmente soltanto una piccola parte di *tweet* per ogni cluster. Avendo raccolto 14 milioni di *tweet*, la sfida principale per gli autori era costituita dal dover etichettare una così grande mole di dati: da questo è scaturita la necessità di utilizzare algoritmi di *clustering* per ridurre la dimensionalità dei dati.

3. **Predizione di tweet ham affidabile**, ovvero utilizzo di algoritmi di *machine learning* addestrati secondo il paradigma del "*learning from positive examples only*" per determinare i *tweet ham*, genuini.

4. **Predizione delle etichette mediante algoritmo EM,** *Expectation-Maximization*

Il loro lavoro è stato la principale fonte di ispirazione per lo sviluppo di questo elaborato e per il sistema qui proposto. Tuttavia, il *dataset* da loro raccolto risale al 2015, tre anni fa, per cui molti dei dati da loro raccolti non sono più disponibili. Inoltre, alcuni passaggi e alcune scelte da loro effettuate non sono ben chiare, per cui, non conoscendo le motivazioni che li hanno portati a compierle, sono spesso state scelte altre strade.

Un altro dei metodi possibili per analizzare lo *spam* è quello dell'analisi degli URL messi in *blacklist* da servizi come Google SafeBrowsing: Grier et Al. [10], ad esempio, hanno analizzato 25 milioni di URL provenienti da 200 milioni di *tweet* pubblici ed etichettato 3 milioni di questi come *spam*. Hanno, inoltre, scoperto che un qualunque

servizio di *blacklisting* impiega dai 4 ai 20 giorni per rilevare un URL malevolo, sebbene circa il 90% delle vittime visiti l'URL entro due giorni. Di conseguenza, approcci basati sul solo *blacklisting* degli URL non sono sufficienti per l'identificazione repentina degli *spammers*.

Altre tecniche di rilevazione, invece, fanno utilizzo di *honeypots*: ad esempio, Lee et Al. [11], mediante *honeypot*, sono riusciti a rilevare circa 2000 profili di *spammer* fra MySpace e Twitter. La classificazione ha fatto leva su quattro caratteristiche: connessioni dell'utente, caratteristiche dell'attività dell'utente, contenuti a cui ha contribuito l'utente, demografica dell'utente.

Sono state proposte anche tecniche basate sulle relazioni sociali degli utenti. Song et Al. [12] propongono due caratteristiche per analizzare il grafo sociale degli utenti: distanza e connettività. Distanza è la lunghezza del percorso più corto fra due utenti; la connettività è misurata mediante *min-cut* e *random walk*. Questo lavoro ha portato a due importanti scoperte: la maggior parte dello spam proviene da utenti a distanza massima di 3; la connettività fra *spammer* e *non-spammer* è diversa da quella fra soli *non-spammer*. Il loro lavoro, tuttavia, presenta due grandi limitazioni: messaggi da nuovi utenti sono classificati come *spam*; messaggi di *spam* da utenti compromessi potrebbero essere etichettati come messaggi benigni.

Per questo motivo, nell'ambito della ricerca, particolare attenzione è posta anche sugli account compromessi.

In Egele et Al. [13], ad esempio, viene introdotto COMPA, un sistema automatico di identificazione di account compromessi. Questo sistema sfrutta la storia recente di ogni utente per costruire un modello comportamentale ed analizzare rispetto a questo tutti i nuovi *tweet*. Il modello comportamentale è costituito di diverse *features*, fra cui gli orari in cui solitamente si è attivi, le applicazioni usate per inviare *tweet*, la lingua utilizzata, gli argomenti solitamente trattati nei propri *tweet*. Da ogni tweet vengono estratte le stesse *features* e confrontate col modello comportamentale costruito per lo stesso utente per ottenere un indice di anomalia del *tweet*. Se questo indice supera una certa soglia, allora l'account potrebbe essere compromesso. Chiaramente, un singolo utente può improvvisamente cambiare abitudini o sperimentare nuove possibilità, per cui gli utenti normali sono solitamente divisi in gruppi, "campagne". Una campagna risulta compromessa se la maggioranza dei suoi utenti è compromessa.

Questo sistema, tuttavia, presenta diverse limitazioni: un attaccante che è a conoscenza della presenza di COMPA potrebbe postare messaggi che si allineano quanto più possibile col profilo comportamentale dell'utente compromesso; inoltre, le misure di similarità utilizzate per il raggruppamento degli utenti non tengono conto, ad esempio, degli URL espansi, per cui utenti compromessi allo stesso modo potrebbero non essere raggruppati nella stessa campagna.

Infine, alcuni lavori realizzati nel laboratorio NdsLab dell'Università degli Studi di Palermo, svolti dai gruppi di ricerca coordinati dai Proff. Gaglio e Lo Re, e relativi allo *spam detection* su Twitter includono un sistema di *topic detection* per individuare *tweet* che fanno riferimento ad emergenze di sicurezza, dovute al rilascio di nuovi *malware* [14] ed un sistema di individuazione *real-time* di *hot topic*, utilizzando una ricerca dinamica di *keyword* all'interno di finestre temporali di dimensione variabile [15][16].

CAPITOLO 2: I Social Network

I *social network* sono oggi diventati una delle maggiori fonti di informazione per le persone. È stato mostrato, inoltre, come questi possano essere usati come strumento per la predizione di indici finanziari ed elezioni politiche.

L'aumento dell'importanza dei social network ha portato un numero sempre più grande di utenti malevoli, *spammer*, ad utilizzarli per i loro scopi, ovvero postare informazioni irrilevanti, false e potenzialmente dannose per gli altri utenti.

Una nuova categoria di *spammer* è quindi emersa, i *social spammer*. A differenza degli *spammer* tradizionali, che utilizzano false *email* o *link* malevoli, i *social spammer* hanno bisogno di attrarre persone e far sì che queste diventino loro *followers*: per questo motivo, spesso, i *social spammer* si fingono utenti comuni o *hub* di informazioni, postando *spam* reale di tanto in tanto.

Nel seguito viene introdotto Twitter, il *social network* di interesse in questo elaborato e vengono esposte alcune delle problematiche ad esso legate incontrate durante lo svolgimento del lavoro.

2.1 – Twitter

Twitter è, ad oggi, uno dei *social network* più famosi ed utilizzati. Creato nel 2006, conta ad oggi oltre 500 milioni di utenti e produce un fatturato di circa 34 miliardi di dollari.

Fornisce ad ogni utente iscritto una pagina personale su cui poter scrivere (o cinguettare, in inglese *to tweet*) messaggi di testo di 280 caratteri al massimo. I messaggi inviati vengono mostrati nella pagina personale dell'utente e comunicati a tutti gli utenti che si sono iscritti per riceverli, ovvero i *followers*.

Sebbene Twitter non sia la piattaforma più popolare in termini di utenti attivi mensilmente, come mostrato in *Figura 1*, la sua popolarità in ambito accademico e di ricerca è elevata: la maggior parte degli studi, infatti, si concentra proprio su Twitter.

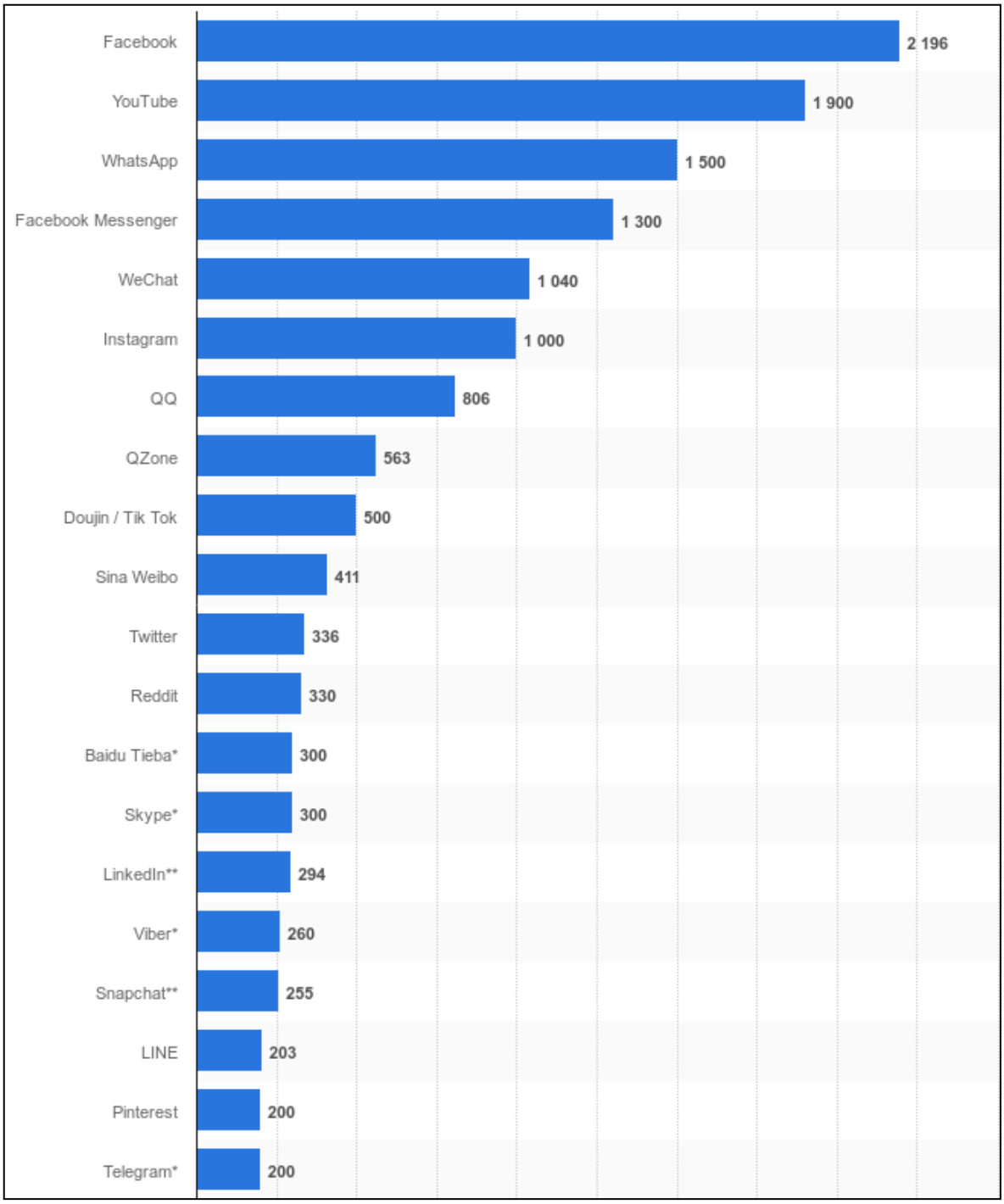

***Figura 1**: Numero, in milioni, di utenti attivi mensilmente sui principali social media. Dati prelevati da statista.com [25] e relativi al mese di Luglio 2018. Come possibile notare, Twitter non è neanche fra i primi 10, essendo solo undicesimo, ma ciò non influisce sulla sua popolarità in ambiente accademico e di ricerca.*

Il motivo di questo successo è da ricercarsi nella sua infrastruttura e nella disponibilità dei dati: nessun altro *social media*, infatti, fornisce gratuitamente quasi il 100% dei suoi dati tramite API pubbliche.

2.1.1 - Panoramica API

Twitter fornisce accesso programmatico ai suoi dati tramite API, *Application Programming Interface*. Per accedere ai dati ed interagire con le API, viene richiesto agli utenti di registrare un'applicazione sulla piattaforma Twitter Apps.

Le API sono organizzate in tre famiglie: *standard*, *enterprise* e *premium*.

- **Standard**: disponibili gratuitamente tramite semplice registrazione della propria applicazione. Consistono di API REST e API Streaming
- **Enterprise**: prevedono un abbonamento a pagamento, forniscono maggiori strumenti per effettuare analisi sui dati
- **Premium**: prevedono un pagamento man mano che si utilizzano, forniscono una versione migliore in termini di affidabilità e robustezza delle API Enterprise

Per quanto riguarda l'accesso ai dati, questo avviene attraverso richieste ad *endpoint*, indirizzi che corrispondono ad uno specifico tipo di informazione fornita. Le API Twitter prevedono un gran numero di *endpoint*, ma questi possono essere organizzati in 5 gruppi principali:

- **Account ed Utenti:** permettono di gestire il profilo di un account, bloccare altri utenti, gestire followers, richiedere informazioni sull'attività di un account autorizzato e molto altro
- **Tweet e Risposte:** permettono di pubblicare *tweet*, rispondere ad altri *tweet*, cercare *tweet* tramite parole chiave o richiedere *tweet* di determinati utenti
- **Messaggi Diretti:** permettono di gestire i messaggi diretti con altri utenti e di creare esperienze di dialogo basate su *chatbot* o interazione umana
- **Ads:** permettono ad aziende esterne di creare e gestire campagne pubblicitarie su Twitter, identificando argomenti di interesse e fornendo utili strumenti per la gestione delle stesse campagne
- **Strumenti per Sviluppatori**: permettono l'*embed* di contenuti come tasti di condivisione su altre pagine *web*. Facilitano la condivisione di informazioni e articoli sui siti *web*.

Endpoint	Descrizione	Richieste/Finestra	Max Dati/Richiesta
/application /rate_limit_status	Consente di ottenere informazioni sullo stato delle richieste effettuate e rimanenti per ogni altro *endpoint*	180	-
/followers/ids	Consente di ottenere gli ID dei *followers* di uno specifico utente, dal più recente al meno recente	15	5000
/statuses /user_timeline	Consente di ottenere la *timeline* di uno specifico utente. Non permette di recuperare più di 3200 *tweet*	900	200
/search/tweets	Consente di accedere allo *stream* di *tweet* pubblici	180	100
/users/show	Consente di ottenere informazioni su uno specifico utente, come il numero di *followers* o il numero di liste pubbliche a cui l'utente è iscritto	900	-

***Tabella 1**: Endpoint più utilizzati durante l'estrazione dei dati, loro descrizione e relativi limiti posti da Twitter per l'interazione con questi.*

Una delle principali problematiche legate all'interazione con le API è costituita dal meccanismo di temporizzazione delle richieste. Twitter, infatti, prevede finestre temporali di 15 minuti, durante le quali è possibile effettuare un numero limitato di richieste ad ogni *endpoint*. Al termine di una finestra temporale tutti i limiti vengono resettati ed è possibile riprendere ad effettuare le richieste.

In *Tabella 1*sono riportati gli *endpoint* più utilizzati durante la costruzione del *dataset*, con relativi limiti.

***Figura 2**: Esempio di tweet pubblicato da Twitter Dev, account ufficiale degli sviluppatori Twitter.*

2.1.1.1 - Formato dei Messaggi Scambiati

Tutte i messaggi di richiesta e risposta scambiati con le API contengono dati codificati in JSON, *JavaScript Object Notation*.

JSON è un linguaggio basato su coppie chiave-valore, con attributi aventi un nome specifico ed un valore associato. Questi attributi permettono di descrivere oggetti.

Twitter fornisce la maggior parte degli oggetti, come *tweet* ed utenti, in JSON. Questi oggetti sono incapsulati e contengono degli attributi che li descrivono. Per esempio, ogni tweet ha un autore, un testo, un ID, un *timestamp* ed altro. Ogni utente ha un nome, un ID, un numero di *followers* ed altro.

Inoltre, ogni *tweet* contiene anche degli oggetti *entity*, ovvero *array* di contenuti comuni su Twitter, come *hashtag*, menzioni, *media*, *link*. Nel caso di *link*, inoltre, nel *payload* di JSON è possibile identificare metadati come l'URL espanso, il titolo e la descrizione della pagina *web*. In totale, oltre 150 attributi possono essere associati ad un singolo *tweet*.

In *Figura 2* è riportato un esempio di *tweet*, in *Figura 3* il corrispettivo messaggio in JSON.

```
{
  "created_at": "Thu Apr 06 15:24:15 +0000 2017",
  "id_str": "850006245121695744",
  "text": "1\/ Today we\u2019re sharing our vision for the future of the Twitter API
    platform!\nhttps:\/\/t.co\/XweGngmxlP",
  "user": {
    "id": 2244994945,
    "name": "Twitter Dev",
    "screen_name": "TwitterDev",
    "location": "Internet",
    "url": "https:\/\/dev.twitter.com\/",
    "description": "Your official source for Twitter Platform news, updates & events.
      Need technical help? Visit https:\/\/twittercommunity.com\/ \u2328\ufe0f
      #TapIntoTwitter"
  },
  "place": {
  },
  "entities": {
    "hashtags": [
    ],
    "urls": [
      {
        "url": "https:\/\/t.co\/XweGngmxlP",
        "unwound": {
          "url": "https:\/\/cards.twitter.com\/cards\/18ce53wgo4h\/3xo1c",
          "title": "Building the Future of the Twitter API Platform"
        }
      }
    ],
    "user_mentions": [
    ]
  }
}
```


***Figura 3**: Codice JSON relativo al tweet mostrato in Figura 2. Come possibile notare, ogni tweet contiene informazioni sulla data di creazione, id, testo e luogo da cui è stato inviato, se la geolocalizzazione è attiva. Inoltre, ogni tweet contiene un oggetto utente, relativo all'autore del tweet stesso, e molte informazioni relative a questo, come id, nome, screen name e descrizione. Infine, sono contenute le entità, insiemi di oggetti comuni su Twitter, come URL, menzioni ed hashtag. Nel caso del tweet in questione, i campi hashtag e user_mentions sono vuoti in quanto il tweet originale non conteneva alcun hashtag né menzione.*

2.1.2 - Condivisione Dati e Privacy

Le *policy* di Twitter pongono un grande numero di restrizioni sull'uso delle API, dei dati ottenibili da queste e sulla loro diffusione.

In particolare:

- Non è possibile in alcun modo utilizzare i dati geografici se non per identificare il luogo da cui un *tweet* è stato mandato
- Non è possibile condividere dati relativi ad utenti o *tweet*, se non i loro identificativi
- Esistono restrizioni sul numero di *tweet* condivisibili per utente, per giorno
- Chiunque scarichi i dati, deve accettare tutti i termini d'uso e le *policy* per gli sviluppatori.

Queste policy, molto stringenti, sono la causa dell'assenza di dataset completi di *tweet* online. Qualunque *dataset*, infatti, fornisce una quantità abbastanza limitata di dati, per cui tutte le analisi richiedono prima una fase di raccolta dei dati.

2.1.3 - Spam su Twitter

Come già anticipato, la diffusione dei *social network* ha portato ad un incremento di *spammer* su questi, oltre che alla nascita dei *social spammer*, utenti il cui unico obiettivo è quello di conquistare quanti più *followers* possibili.

Il problema dello *spam* su Twitter è diventato rilevante a partire dalle elezioni presidenziali negli Stati Uniti del 2016: in quell'occasione, infatti, si è scoperto l'utilizzo di falsi account russi in grado di influenzare l'opinione pubblica e, di conseguenza, i risultati delle elezioni [24]. Twitter fu accusata di aver permesso la diffusione di questi account senza alcun tipo di contromisura efficace [26].

A partire da quel momento, diversi sono stati gli sforzi di Twitter per combattere il problema dello *spam*: solo nel mese di giugno 2018 la società ha annunciato l'acquisizione di Smyte [27], *startup* di Sicurezza Informatica che offre strumenti per combattere lo *spam* e proteggere gli account degli utenti, ed ha inoltre reso necessaria l'autenticazione di ogni *account* mediante *email* o numero di cellulare.

Inoltre, la stessa Twitter sta sviluppando un algoritmo di *machine learning* in grado di trovare *account* potenzialmente malevoli, prima ancora che questi vengano segnalati manualmente. Secondo dati recenti, questo algoritmo ha identificato una media di quasi 10 milioni di casi di potenziali *spammer* a settimana nel solo mese di maggio 2018.

Inoltre, Twitter ha adottato il servizio di *blacklisting* di URL di Google SafeBrowsing, per l'individuazione rapida di *tweet* malevoli.

Tuttavia, l'efficacia di questo metodo è abbastanza ridotta, come fatto notare da Grier et Al. [10], i quali hanno studiato i sistemi di *blacklisting* e notato come la maggior parte del traffico su un qualunque *link* arrivi nei primi due giorni di vita, mentre i sistemi di *blacklisting* ne impieghino almeno quattro per identificare un sito malevolo.

Infine, Twitter ha ridotto la visibilità degli *account* sospetti: su questi viene applicato un segnale di *warning* e nessun nuovo utente può seguirli. Lo stato di "sospetto" può essere rimosso mediante il superamento di una "sfida", ad esempio la verifica di un numero di cellulare.

CAPITOLO 3: Il Sistema Semiautomatico di Annotazione

Dati tutti i problemi prima evidenziati, tra cui la necessità di avere un *dataset* etichettato e l'indisponibilità di dati più vecchi di qualche anno, si è reso necessario costruire un *dataset* nuovo, catturando dati in *real-time* tramite API di Twitter.

Sui dati raccolti si è poi proceduto ad effettuare un'analisi in più passi successivi, con lo scopo di etichettare in maniera quanto più automatica possibile gli utenti, distinguendoli fra *spammer* e genuini.

Nel seguito viene illustrato nel dettaglio come si sono svolte le due fasi di costruzione ed analisi, e vengono introdotti, dove necessario, particolari algoritmi dello stato dell'arte utilizzati.

3.1 - Costruzione del Dataset

Per poter ottenere dati da Twitter è necessario possedere un *account* sul *social network* in questione, aggiungere un numero di cellulare all'account e registrare il programma che richiede accesso alle API sulla relativa piattaforma, Twitter Apps. Fatto ciò, Twitter garantisce l'accesso alle API e la possibilità di effettuare richieste.

La raccolta effettuata si è articolata in due fasi: ricerca ed espansione.

Le due fasi sono procedute parallelamente, svolte da due *thread* concorrenti sincronizzati mediante il paradigma del produttore/consumatore, noto nell'ambito della programmazione concorrente.

In questo caso, il *thread* di ricerca, che svolge il ruolo del produttore, aggiunge *tweet* ad una lista condivisa di *tweet* da espandere, mentre il *thread* di espansione preleva i *tweet* stessi e li espande.

In *Figura 4* è mostrato l'intero processo di costruzione del dataset, che viene spiegato nel dettaglio nei prossimi due paragrafi.

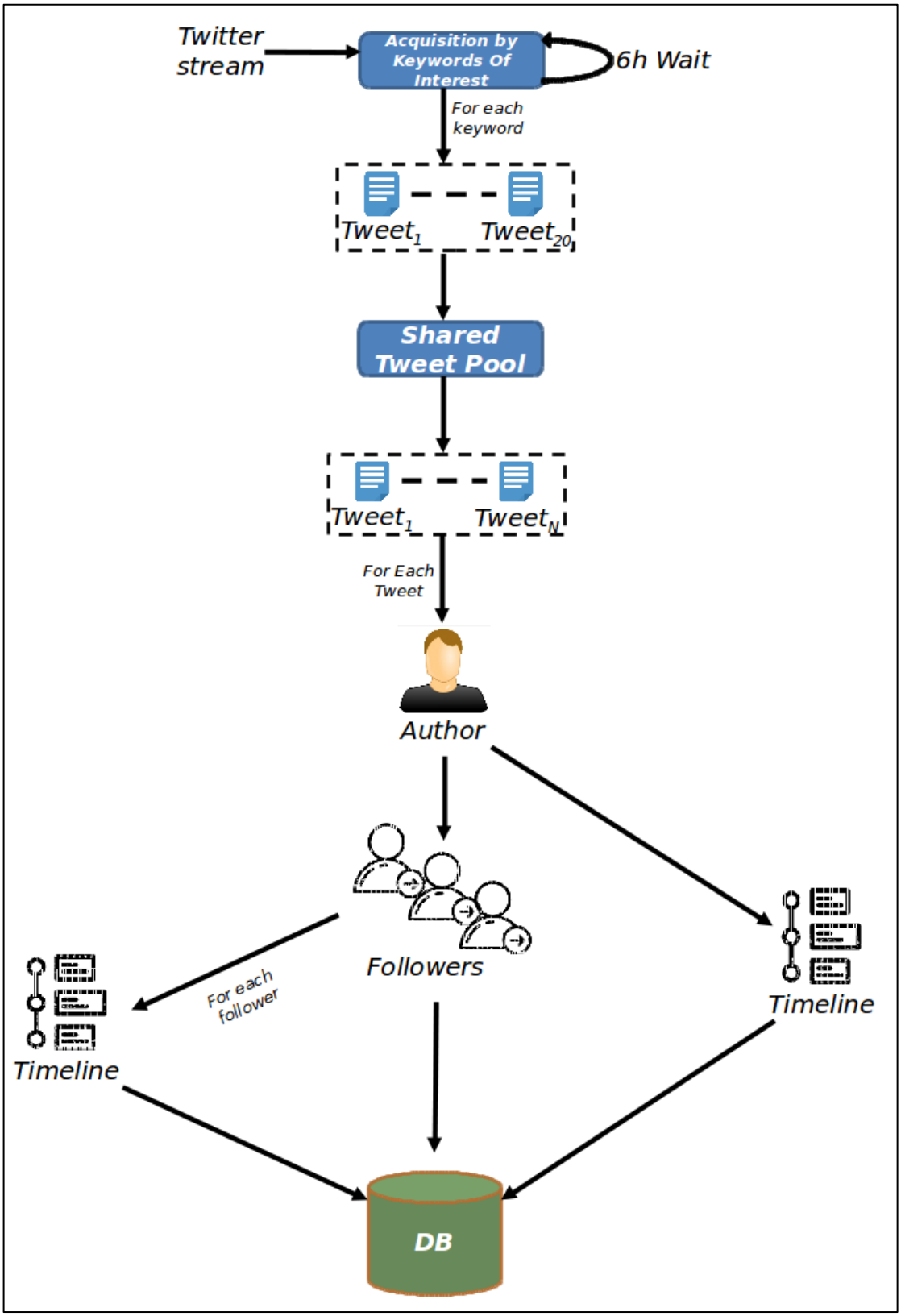

Figura 4: *Immagine riassuntiva del processo di costruzione del dataset. In alto, la fase di ricerca che seleziona keywords e raccoglie tweet. Al centro e in basso, la fase di espansione dei tweet: per ogni tweet vengono raccolti autore, timeline, followers e timeline di questi. Tutti i dati sono quindi memorizzati nel DB.*

3.1.1 - Ricerca

La ricerca, ovvero la selezione di nuovi *tweet* da espandere, è andata avanti ad intervalli regolari: ogni 6 ore sono state selezionate delle parole chiave, *keywords*, con cui effettuare richieste alle API Twitter.

Obiettivo della ricerca è quello costruire un *dataset* quanto più bilanciato possibile fra utenti *spammer* e genuini, nonostante gli *spammer* siano in numero molto minore e più difficili da individuare.

Come suggerito in Sedhai et Al [9], un'opportuna scelta delle *keywords* può portare ad individuare *tweet* di *spam* con una maggiore probabilità. In particolare, gli autori suggeriscono di utilizzare *hashtag* popolari e parole legate ad alcune tematiche come contenuti per adulti e guadagno di denaro.

Prendendo spunto da questo modello, ad ogni iterazione della ricerca sono state selezionate 5 *keywords*, così distribuite:

- **3 estratte randomicamente dai "trending topic" di Twitter**. I *trending topic* sono gli argomenti più discussi su Twitter per ogni dato istante per ogni luogo nel mondo. Le API consentono accesso ai *trending topic* di tutto il mondo ("worldwide") oppure a quelli di un luogo in particolare, specificato mediante il codice WOEID, *Where On Earth ID*, di Yahoo. Dato che i *trending topic worldwide* contengono spesso *keywords* in alfabeti diversi da quello latino, si è scelto di utilizzare soltanto i *trending topic* degli USA

- **2 estratte a caso dall'elenco di parole "spammy"** identificate da Sedhai et Al. [9]. Queste parole si presentano con maggiore probabilità in un *tweet* di *spam* piuttosto che in un *tweet* genuino. Alcune di queste sono riportate in *Tabella 2*

- Opzionalmente, durante il mese di luglio 2018, la parola chiave **"World Cup"** è stata aggiunta, per cercare di catturare in tempo reale i *tweet* relativi al Mondiale di Calcio 2018

Durante ogni iterazione della ricerca, per ogni *keyword* sono stati estratti 500 *tweet*, corrispondenti al limite massimo consentito dalle API per la ricerca di *tweet*. Di questi, soltanto 20 per *keyword* sono stati estratti e sono passati alla fase successiva, quella di espansione.

"Spammy" Words		
500aday	maxvip	myretweets
androidgames	openfollow	mustfollow
autofollow	ipad	android
followback	rt2club	hitfollow

***Tabella 2**: Alcune delle keywords identificate come spammy in Sedhai et Al. [9]. Secondo gli autori, queste parole hanno una maggior probabilità di trovarsi in tweet di spam piuttosto che in tweet genuini.*

3.1.2 - Espansione

Per espansione di un *tweet* si intende l'acquisizione di un insieme di dati legati al *tweet* in questione. In particolare, le informazioni di interesse che sono state raccolta a partire da ogni *tweet* sono:

- **Informazioni sul tweet corrente**: ID, ID dell'utente creatore, data di creazione, testo, lingua, eventuali *hashtag* presenti, eventuali menzioni presenti, eventuali URL presenti e fonte, ovvero applicazione da cui il *tweet* è stato inviato
- **Informazioni sull'utente creatore del tweet**: ID, *screen name*, descrizione, numero di *followers*, numero di amici, data di creazione dell'account, numero di *tweet* inviati, lingua, informazione sulla verifica dell'account, *timezone* e *offset* UTC
- **Timeline del creatore del tweet**, ovvero i 200 *tweet* più recenti inviati dall'utente in questione
- **Followers del creatore del tweet**, ovvero 20 utenti a caso fra quelli che seguono l'utente in questione
- **Timeline di ciascuno dei followers** selezionati al passo precedente

Tutte queste informazioni sono state quindi memorizzate nel *database*. Inoltre, è stata anche memorizzata la relazione di *following* fra utente seguito ed ogni suo *follower*.

3.2 - Algoritmi per l'annotazione del dataset

Ottenuto un *dataset* su cui poter lavorare, si è potuto procedere con un'analisi dello stesso, con lo scopo di etichettare gli utenti raccolti, distinguendo utenti *spammer* da utenti genuini, cercando di massimizzare il numero di utenti etichettati automaticamente e minimizzare il numero di utenti da etichettare in modo manuale.

*** OMISSIS ***

3.2.1 - Analisi degli URL

L'analisi degli URL, che costituisce la prima fase del processo di analisi del *dataset*, porta ad una prima etichettatura degli utenti. Come suggerisce il nome stesso, l'etichettatura dipende da un'analisi basata sui soli URL condivisi da ogni utente, senza tenere in considerazione altro tipo di informazioni.

*** OMISSIS ***

3.2.2 - Near Duplicates Clustering

*** OMISSIS ***

3.2.2.1 - Tweet Tokenization

Come prima anticipato, un fondamentale passo di *pre-processing* dei tweet, prima di sottoporli all'applicazione di MinHash e LSH, è costituito dalla Tweet Tokenization, tokenizzazione dei tweet, ovvero il processo di estrazione di *token*, parole significative, a partire da un *tweet* e dal suo testo.

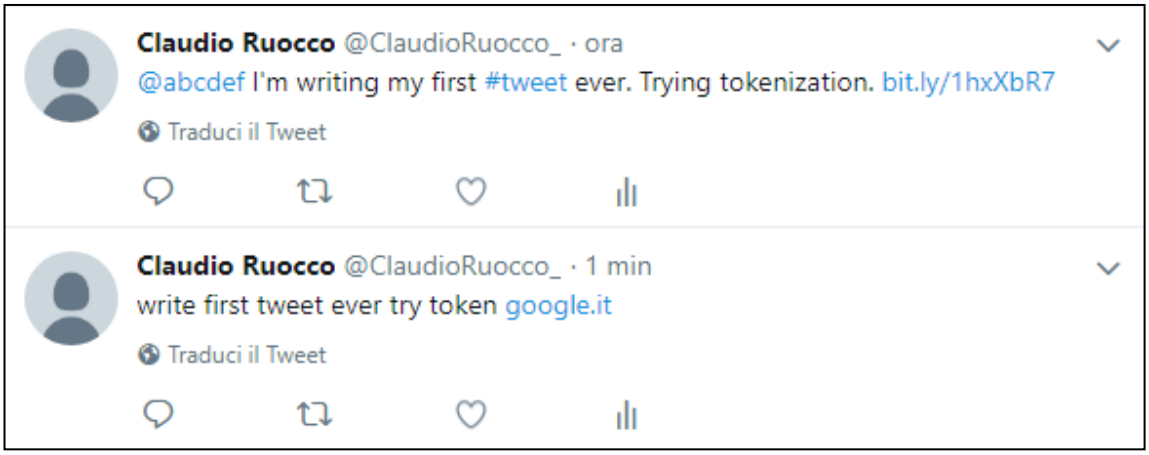

***Figura 6**: In alto, esempio di tweet da tokenizzare. In basso, corrispondenti token individuati applicando l'algoritmo di tokenizzazione. Come possibile notare, la menzione è stata rimossa, così come il simbolo "#" prima della parola tweet, l'URL è stato espanso e le altre parole sono state sottoposte a stemming. Il risultato finale del processo di tokenizzazione, dunque, è la lista dei token presenti nella parte inferiore dell'immagine.*

La tokenizzazione è stata articolata nei seguenti passi:

- **Eliminazione di tutti i tweet non in lingua inglese**.
- **Rimozione delle menzioni**.
- **Conversione del testo in lower case.**
- **Applicazione dello Stemming.**
- **Rimozione del carattere "#" dagli hashtag.**
- **Eliminazione di qualunque simbolo.**
- **Espansione degli URL.**
- **Rimozione delle stop-words.**
- **Normalizzazione dei caratteri accentati.**

Un esempio completo di applicazione della tokenizzazione è riportato in *Figura 6*.

Lucene's Stop-Words			
a	an	their	that
they	this	will	to
into	then	was	of
but	are	with	if

***Tabella 3**: Alcune delle stop-words identificata in Apache Lucene. L'elenco completo è disponibile su github.com [28].*

3.2.2.2 - Stemming

Lo *stemming* è uno dei metodi più utilizzati nell'ambito dell'Information Retrieval. Il suo utilizzo permette di raggruppare parole diverse fra loro, ma accomunate dalla stessa forma radice (tema). È ad oggi applicato dai motori di ricerca per l'espansione di interrogazioni ed il reperimento di contenuti di interesse.

Esistono diversi algoritmi di *stemming*, ognuno dei quali presenta diverse regole di sostituzione dei caratteri per giungere ad una forma radice ed ognuno dei quali si può applicare solo su testi scritti in una determinata lingua.

Fra i vari algoritmi, quello più noto in letteratura, e qui utilizzato, è l'algoritmo di Porter, valido per i soli testi in lingua inglese. Alcune delle regole di sostituzione implementate da questo algoritmo sono riportate in *Tabella 4*. Inoltre, una sua implementazione in Java è presente all'interno del *toolkit* ApacheNLP, *Natural Language Processing*.

In *Figura 7* è possibile osservare un esempio di applicazione di *stemming* su alcune parole che condividono la stessa radice, *comput-*.

Oltre alla già anticipata dipendenza dalla lingua, lo *stemming* presenta ulteriori criticità:

- Può accomunare alla stessa forma radice parole con significati molto diversi. Ad esempio, in italiano, *biscia* e *biscotto* potrebbero essere accomunati alla stessa radice, *bisc-*.

Regola di Sostituzione	Esempio di Applicazione
SSES --> SS	caresses -> caress
IES -> I	ponies -> poni
S ->	cats -> cat
EED -> EE	agreed -> agree

***Tabella 4**: Alcune delle regole di sostituzione presenti all'interno dell'algoritmo di stemming di Porter e relativo esempio di applicazione. È bene notare che le regole di sostituzione, nell'implementazione completa, hanno un rigido ordine di applicazione e, molto spesso, delle precondizioni necessarie affinchè si possano applicare.*

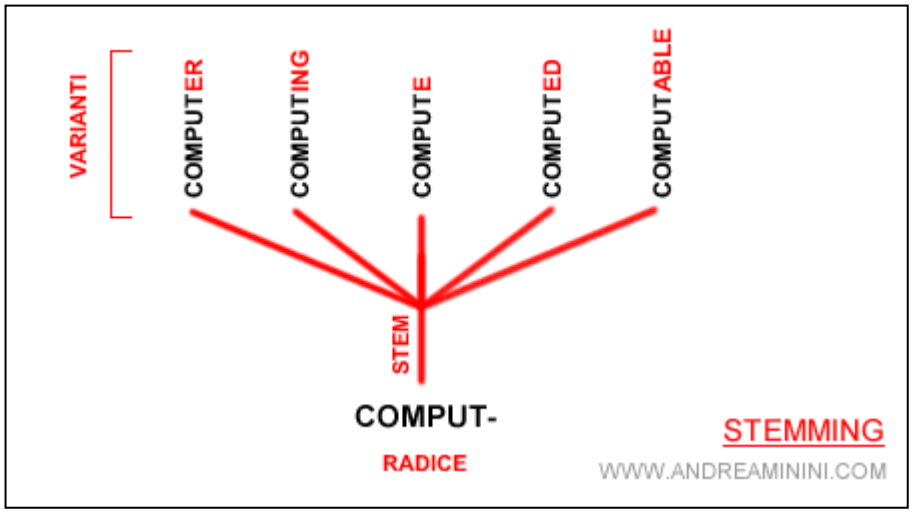

***Figura 7**: Esempio di applicazione di stemming alle parole computer, computing, compute, computed e computable. Tutte queste vengono ricondotte alla stessa forma radice, comput-. Immagine prelevata da andreaminini.com [29].*

- Non riesce ad accomunare parole con radici diverse ma significati simili. Ad esempio, *biscia* e *serpe,* le cui radici sono rispettivamente *bisc-* e *serp-*.
- Non riesce ad identificare correttamente radici per parole composte, come pianoforte.

A causa dei motivi prima elencati, lo *stemming* è spesso accompagnato da lemmatizzazione, un processo di analisi morfologica che identifica accuratamente un lemma per ogni parola. Tuttavia, questo processo richiede una complessità computazionale molto più elevata rispetto allo *stemming* e porta a benefici molto limitati per testi molto corti, come i *tweet*. Dunque, nella fase di tokenizzazione dei tweet, ci si è limitati al solo utilizzo dello *stemming*.

3.2.2.3 - URL Expansion

Gli URL, Uniform Resource Locator, sono sequenze di caratteri che identificano univocamente l'indirizzo di una risorsa presente in *internet*.

Molto spesso, grazie all'utilizzo di servizi esterni, detti di URL *shortening*, gli URL si presentano in una versione "*shortened*", accorciata, che nasconde l'URL reale fino a quando non viene cliccata e non si innesca un meccanismo di redirezione fino all'URL finale. Alcuni dei servizi di URL *shortening* più famosi [30][31] sono riportati in *Tabella 5*.

In particolare, in Twitter gli URL *shortened* costituiscono la quasi totalità degli URL presenti. I motivi principali sono due:

1) **Twitter fornisce un servizio di URL shortening**, *t.co*, che rimpiazza automaticamente qualunque URL postato sul *social network* in questione con una sua versione *shortened*.

2) **Ogni tweet ha un limite massimo di 280 caratteri**. Alcuni URL sono particolarmente lunghi in termini di caratteri, per cui molto spesso l'utente è obbligato ad utilizzare un servizio di URL *shortening* per poter inviare un *tweet*.

Non essendo in alcun modo significativa l'analisi su URL *shortened*, si è dunque reso necessario implementare un algoritmo di espansione degli URL, un algoritmo in grado di sostituire ogni URL *shortened* con la sua corrispondente versione espansa.

URL Shorteners più utilizzati			
bit.ly	goo.gl	sumo.ly	shorte.st
is.gd	mcaf.ee	tinyurl.com	moourl.com
bit.do	adf.ly	ow.ly	deck.ly
t.co	wp.me	su.pr	x.co

Tabella 5*: Alcuni dei servizi di URL shortening più diffusi in rete. Alcuni di questi sono stati individuati mediante statista.com [30], altri mediante alcuni articoli presenti in rete, fra cui quello di lifewire [31].*

*** OMISSIS ***

3.2.2.4 - MinHash, LSH

Il MinHash è una tecnica che consente di stimare rapidamente quanto simili siano due insiemi di dati. È stato inizialmente usato nei motori di ricerca per rilevare pagine *web* duplicate ed eliminarle dai risultati di ricerca. Trova, tuttavia, la sua applicazione principale nei problemi di *clustering* di documenti ed individuazione di contenuti *near-duplicates*.

Fornisce una rapida approssimazione della similarità di Jaccard di due documenti, ovvero del rapporto fra la cardinalità dell'insieme intersezione e la cardinalità dell'insieme unione delle parole dei due documenti. Questa misura non viene utilizzata per l'identificazione di documenti potenzialmente duplicati in quanto necessita l'applicazione su due soli documenti alla volta. Sarebbe necessario applicarla a tutte le possibili coppie di documenti per riuscire ad identificare i duplicati.

Il MinHash riesce a stimare rapidamente questa misura, sebbene utilizzato da solo non riduca il numero di confronti necessari per identificare contenuti *near-duplicates*. Per questo, è spesso accompagnato da tecniche di LSH.

Document 1
"Hello World, I'm John"

Document 2
"World Hello, I'm Nathan"

Words	H1( )	H2 ( )	H3( )
Hello	1	5	25
World	3	21	15
John	4	12	7
Nathan	7	1	13

MH	H1 ( )	H2 ( )	H3( )
Document 1	1	5	7
Document 2	1	1	13

Numb Of Common Words: 2
Numb Of Total Words: 4

Numb Of Matches in Sig: 1
Dim Of Sig: 3

True Jaccard Sim: 2 / 4 = 0,5
Approx Jaccard Sim: 1 / 3 = 0,333

Figura 9*: Esempio di calcolo della firma MinHash su due documenti e della loro similarità di Jaccard reale ed approssimata. In alto, calcolo dei valori di hash per ogni singola funzione di hash, per ogni token. Al centro, calcolo delle firme minhash dei due documenti. In basso, calcolo di similarità di Jaccard reale ed approssimata.*

In letteratura esistono diverse varianti del MinHash. Le varianti differiscono sia per numero di funzioni di *hash* utilizzate, dato che alcune utilizzano una sola funzione di *hash* ed altre ne utilizzano molteplici, sia per come suddividono il documento. Alcune varianti, infatti, dividono i documenti in sequenze di *k* caratteri consecutivi, altre invece considerano le singole parole (o loro sequenze) di cui questi sono costituiti. La scelta effettuata in quest'elaborato è stata quella di utilizzare più funzioni di *hash*, 200, e di considerare i *tweet* come divisi in singole parole.

Le funzioni di *hash* sono tutte funzioni del tipo $h(x)=(a+bx) \bmod p$, dove *a* e *b* sono i parametri che identificano la singola funzione di *hash*, *x* è un accumulatore, inizializzato a 0, e *p* è un parametro dell'algoritmo.

Le parole vengono processate carattere per carattere: per ogni carattere viene calcolato lo *XOR* fra questo ed il valore presente nell'accumulatore, quindi viene calcolato *h(x)* e l'accumulatore è aggiornato al valore calcolato in *h(x)*. Il valore finale dell'accumulatore, *x*, è il valore di *hash* della parola.

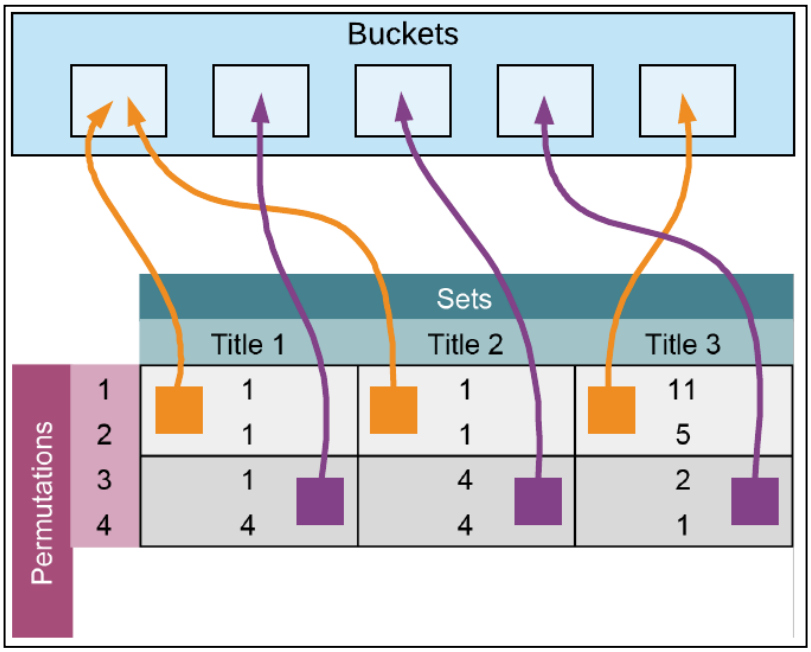

***Figura 10**: Esempio di applicazione di LSH. Le firme MinHash sono divise in bande, i documenti 1 e 2 cadono nello stesso bucket per la prima banda, sono quindi candidati near-duplicates nonostante cadano in bucket diversi per le altre bande.*

In breve, MinHash opera calcolando, per ogni documento, la sua *firma MinHash*, ovvero la sequenza di valori di *hash* minimi ottenuti applicando ogni funzione di *hash* ad ogni *token* di cui è costituito il *tweet*. A partire dalle *firme MinHash*, la similarità di Jaccard approssimata corrisponde al rapporto fra numero di valori coincidenti e numero di valori totali. In *Figura 9* è riportato un esempio completo di applicazione del MinHash e di calcolo delle *firme MinHash* per due diversi documenti.

Come prima anticipato, anche tramite MinHash, tuttavia, il numero di confronti necessari per individuare documenti simili risulta troppo elevato. Per questo motivo, infatti, il MinHash è spesso seguito da approcci di LSH, utili per individuare cluster di documenti simili in maniera più veloce.

LSH, acronimo di *Locality-Sensitive Hashing*, è una tecnica che consente di ridurre le dimensionalità di un insieme di dati. Presa in *input* la matrice di MinHash, ovvero la matrice delle firme MinHash di tutti i documenti, divide questa in "bande" e realizza una hash-table calcolando il valore di hash di ogni documento, per ogni banda, in modo tale che documenti simili ricadano nello stesso *bucket* con alta probabilità per almeno una banda. Se due documenti ricadono nello stesso *bucket* per almeno una banda, allora sono candidati *near-duplicates*.

Differisce quindi dalle normali tecniche di *hash* crittografico perché punta a massimizzare la probabilità di collisione per elementi simili. In *Figura 10* è riportato un esempio di applicazione di LSH.

In seguito all'applicazione di LSH vengono quindi restituiti, per ogni documento, tutti i suoi candidati *near-duplicates*. A causa dell'utilizzo delle funzioni di *hash* per identificare i *near-duplicates*, potrebbero essere presenti dei falsi positivi, documenti candidati come simili ma in realtà differenti fra di loro.

Per questo motivo, generalmente, si utilizza LSH per individuare tutti i candidati *near-duplicates* e si procede poi al calcolo della similarità reale di Jaccard per ogni coppia di candidati *near-duplicates*. Solo quei documenti per cui la similarità di Jaccard risulta superiore ad una certa soglia sono considerati documenti simili.

3.2.2.5 - Features Utilizzate per la Valutazione

*** OMISSIS ***

3.2.3 - Quality Threshold Clustering

*** OMISSIS ***

3.2.3.1 - L'algoritmo: il QTC

Il *Quality Threshold Clustering*, QTC, è un algoritmo di *clustering* di dati, inizialmente introdotto per il partizionamento di geni.

Come tutti gli algoritmi di *clustering*, ha come obiettivo l'identificazione di strutture e *pattern* in dati di dimensionalità elevate. Tuttavia, a differenza degli altri algoritmi di *clustering*, non richiede di specificare a priori il numero di *cluster* da trovare,

per cui assicura buona qualità dei *cluster* risultanti. Inoltre, esecuzioni diverse sugli stessi dati portano all'identificazione degli stessi *cluster*.

Richiede di specificare due parametri:

- **Dimensione Minima**: *cluster* le cui dimensioni non superano questo parametro vengono immediatamente scartati;
- **Diametro Massimo**: l'algoritmo non aggiunge elementi ad un *cluster* se il *cluster* risultante risulta avere un diametro maggiore di quello specificato. Per diametro di un *cluster* si intende la distanza massima fra una qualunque coppia di elementi del *cluster*.

L'algoritmo opera come segue: per ogni elemento da clusterizzare viene costruito un *cluster* candidato aggiungendo di volta in volta al *cluster* quell'elemento che minimizza la crescita del diametro, fino a che non viene raggiunta la dimensione massima consentita per questo. Bisogna notare che, in questa fase, tutti gli elementi sono disponibili per tutti gli altri, anche se inseriti in altri *cluster* candidati.

Conclusa questa fase, viene selezionato fra i *cluster* candidati quello la cui dimensione è massima, ovvero il *cluster* col maggior numero di elementi al suo interno. Se questa dimensione è minore della dimensione minima stabilita, allora il processo termina e tutti gli elementi rimanenti sono da considerarsi come non clusterizzabili.

Altrimenti, tutti gli elementi del *cluster* sono da considerarsi come clusterizzati e vengono rimossi dall'elenco degli elementi da clusterizzare. Il processo, quindi, ricomincia per identificare altri *cluster*.

In *Figura 11* è riportato un possibile pseudocodice per l'algoritmo. La versione riportata è quella inizialmente prevista dagli autori, senza alcun limite inferiore per la dimensione dei *cluster*.

Il maggior svantaggio del QTC rispetto agli altri algoritmi di *clustering*, tuttavia, è legato alla sua complessità computazionale che, secondo le stime di Danalis et Al. [17], è $O(n^3 * F_D)$, dove F_D è la complessità computazionale della funzione utilizzata per calcolare il diametro. La versione originale del QT, proposta da Heyer et Al. [18] prevede una complessità computazionale di $O(n^5)$.

```
Procedure QT_Clust(G, d)
if (|G| ≤ 1) then output G, else do                    /* Base case */
 foreach i ∈ G
     set flag = TRUE; set A_i = {i} /*A_i is the cluster started by i*/
     while ((flag = TRUE) and (A_i ≠ G))
       find j ∈ (G − A_i) such that diameter(A_i ∪ {j}) is minimum
       if (diameter(A_i ∪ {j}) > d)
           then set flag = FALSE
           else set A_i = A_i ∪ {j}          /* Add j to cluster A_i */
 identify set C ∈ {A_1, A_2, .., A_|G|} with maximum cardinality
 output C
 call QT_Clust(G − C, d)
```

***Figura 11**: La funzione QT_Clust che prende a parametro l'insieme di elementi da clusterizzare, G, e il diametro massimo consentito per i cluster, d. Ad ogni chiamata, produce in output il cluster trovato, C, e richiama ricorsivamente sé stessa sul sottoinsieme ottenuto rimuovendo da G tutti gli elementi di C.*

```
Procedure QT_Clust_radius(G, d)
if (|G| ≤ 1) then output G, else do    /* Base case */
  for each i ∈ G
   set flag = TRUE; set A_i = {i} /* A_i is the cluster started by i */
   while ((flag) and (A_i ≠ G))
     for each j ∈ (G − A_i)
     if (distance (i, j) > d/2)
        then set flag = FALSE
        else set A_i = A_i ∪ {j} /* Add j to cluster A_i */
identify set C ∈ {A_1, A_2, ⋯, A_|G|} with maximum cardinality
output C
call QT_Clust_radius(G − C, d)
```

***Figura 12**: Pseudocodice per la variante DrN QT. Come possibile vedere, considera il calcolo di distanze rispetto ad un punto centrale fisso (il "generatore" del cluster) piuttosto che di diametri.*

Questa complessità computazionale così elevata rende l'algoritmo inutilizzabile su grandi quantità di dati: per questo motivo, in letteratura esistono diverse varianti più efficienti di questo algoritmo.

Versione QT	Dataset 1.060 punti		Dataset 10.000 punti	
	Tempo di Esecuzione Medio (in secondi)	**Rapporto di Efficienza**	**Tempo di Esecuzione Medio (in secondi)**	**Rapporto di Efficienza**
Heyer QTC	363	4.84	1.692.545	19.2
Danalis QTC	75	1	88.296	1
DrN QTC	1	0.013	122	0.001

***Tabella 6**: Risultati degli esperimenti di Loforte Jr. [19] in merito all'efficienza delle diverse varianti del QTC. I risultati sono relativi a due dataset pre-etichettati, uno costituito da 1060 punti e l'altro da 10.000 punti. Inoltre, i tempi di esecuzione sono delle medie ottenute da più esecuzioni consecutive. Sul dataset da 1.060 punti, la variante DrN impiega 1 solo secondo per l'esecuzione, rispetto ai 363 richiesti dalla versione originale dell'algoritmo. Sul dataset da 10.000 punti, invece, la variante DrN impiega 122 secondi per effettuare il clustering dei 10.000 punti, mentre la versione originale del QT impiega oltre 1.500.000 secondi, corrispondenti a circa 19 giorni.*

In particolare, la versione utilizzata in questo elaborato è il DrN QT, variante che considera ogni elemento da clusterizzare come "generatore" di un possibile *cluster* ed aggiunge, di volta in volta, solo quegli elementi la cui distanza dal generatore risulta inferiore alla soglia fissata. Abolisce, quindi, il calcolo del diametro massimo, che costituisce il contributo maggiore alla complessità computazionale della versione originale dell'algoritmo. In *Figura 12* è riportato un possibile pseudocodice per questa variante del QT.

Per quanto riguarda l'efficienza, in Loforte Jr. [19] questa viene testata su due *dataset* pre-etichettati, uno costituito da 1060 punti e l'altro da 10.000 punti, ed è calcolata come rapporto fra tempo di esecuzione di una variante di riferimento del QT, ovvero quella proposta in Danalis et Al. [17], e tempo di esecuzione di tutte le altre varianti. In particolare, sul *dataset* da 1060 punti, la versione originale del QT realizza uno *score* di 4.84, mentre la versione DrN QT realizza 0.013. Sul dataset da 10.000 punti, invece, la

versione originale del QT risulta avere un'efficienza di 19.2, mentre il DrN QT ha un'efficienza di 0.001. I risultati dettagliati sono riportati in *Tabella 6*.

3.2.3.1 - Features Utilizzate per la Valutazione

*** OMISSIS ***

CAPITOLO 4: Risultati Sperimentali

Gli algoritmi di *clustering*, come MinHash e QTC, hanno bisogno di una fase di calibrazione dei propri parametri per poter essere utilizzati con risultati soddisfacenti. I risultati, infatti, variano al variare dei parametri degli algoritmi. Per questo motivo, si è resa necessaria una fase di calibrazione degli stessi.

In questo capitolo, viene inizialmente spiegata la necessità della presenza di un *ground-truth* per la valutazione dei risultati, quindi vengono introdotte le principali metriche utilizzate per valutare gli stessi e, infine, vengono esposti i vari esperimenti effettuati per calibrare i parametri del sistema.

Per poter verificare la correttezza di un qualunque algoritmo di etichettatura o di *clustering*, come MinHash e LSH, calcolare metriche e valutare i risultati, è necessario avere un *ground-truth*, un dataset in cui i dati sono già annotati ed utilizzabile come riferimento.

Tuttavia, anche a causa delle stringenti *policy* sulla *privacy* di Twitter, in rete non sono disponibili molti *dataset* già etichettati e pronti all'uso. Due dataset in particolare, tuttavia, sono stati utilizzati, ovvero:

- **Sedhai et Al. [9]:** *dataset* contenente una raccolta di circa 14 milioni di *tweet*, etichettati come *spam* o *ham* (non *spam*). Il dataset risale al 2015, per cui buona parte dei *tweet* e degli utenti risultano non più disponibili; inoltre, le etichette sono assegnate ai *tweet* e non ai corrispettivi utenti, per cui si è proceduto a segnare come *spammer* un utente avente almeno un tweet etichettato come *spam* e come genuino un utente non avente alcun *tweet* di *spam*.

- **Tao et Al. [20]:** *dataset* contenente un insieme di coppie di *tweet* etichettati con un giudizio di similarità da 1 a 5, dove 1 indica similarità minima e 5 similarità massima, ovvero *tweet* duplicati. Come per il precedente *dataset*, anche per questo diverse informazioni non sono più disponibili. Inoltre, non tutte le possibili coppie di *tweet* risultano essere etichettate.

Avendo a disposizione un *ground-truth* ed il risultato dell'applicazione di un qualche algoritmo, per valutare la qualità dei risultati ottenuti sono necessarie delle metriche di valutazione.

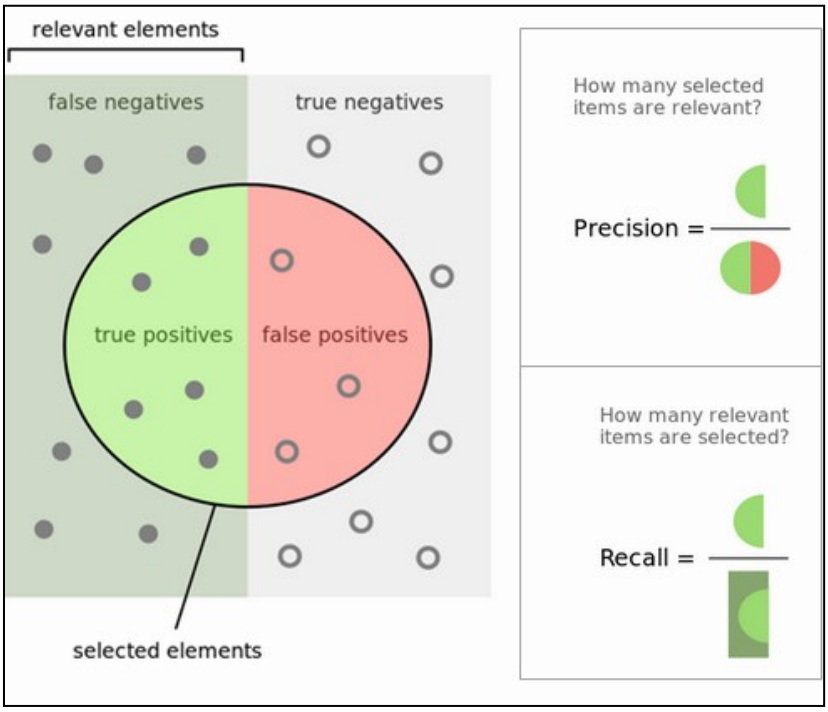

***Figura 13**: A sinistra, all'interno del cerchio, le coppie risultati dall'applicazione di un algoritmo di clustering. Sullo sfondo, le coppie costituenti il ground-truth. A destra, spiegazione del calcolo di due importanti metriche, precision e recall.*

Nel caso di MinHash e LSH, le metriche devono essere in grado di valutare quanto simili siano i *cluster* ottenuti rispetto ai *cluster* di *ground-truth*. Nel caso del QTC, le metriche devono essere in grado di valutare la bontà dei *cluster* ottenuti, in termini di coesione e separazione. Nel seguito, sono descritte nel dettaglio le metriche utilizzate.

- **True Positives** (TP): numero di coppie di elementi presenti sia nel *ground-truth* che nei risultati del *clustering*
- **False Positives** (FP): numero di coppie di elementi non presenti nel *ground-truth* ma risultanti dall'applicazione dell'algoritmo di *clustering*
- **True Negatives** (TN): numero di coppie non presenti né nel *ground-truth* né nei risultati del *clustering*
- **False Negatives** (FN): coppie presenti nel *ground-truth* ma non risultanti dall'applicazione dell'algoritmo di *clustering*

- **Precision**: rapporto fra *true positives* e somma di *true positives* e *false positives*. Indica la frazione di elementi corretti restituiti rispetto alla totalità di elementi restituiti.

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

- **Recall**: rapporto fra *true positives* e somma di *true positives* e *false negatives*. Indica la frazione di elementi corretti restituiti rispetto alla totalità di elementi che l'algoritmo avrebbe dovuto restituire.

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

- **F-Score**: media armonica di *precision* e *recall*, generalmente utilizzata per sintetizzare in un unico valore i risultati delle altre due metriche. Il suo valore varia fra 0 ed 1, dove 0 indica corrispondenza minima ed 1 corrispondenza massima fra i *cluster* di *ground-truth* ed i *cluster* ottenuti dall'applicazione dell'algoritmo di *clustering*.

$$F - Score = \frac{2}{Precision^{-1} + Recall^{-1}}$$

- **Accuracy**: rapporto fra somma di *true positives* e *true negatives* e somma di tutti i parametri. Indica la frazione di predizioni corrette del sistema rispetto a tutte le predizioni effettuate.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

In *Figura 13* è schematizzato il metodo di calcolo di *true positives*, *false positives*, *false negatives* e *true negatives*. Inoltre, sono spiegate le metriche di *precision* e *recall*.

- **WSS**: metrica che misura quanto vicini siano gli elementi di un *cluster*, ovvero la coesione dei *cluster*. È pari alla somma dei quadrati delle distanze di ogni elemento rispetto al centroide del *cluster* di appartenenza.

$$WSS = \sum_{i} \sum_{x \in C_i} (x - m_i)^2$$

- **BSS**: metrica che misura quanto distinti, o ben separati, siano i *cluster* tra loro. È pari alla somma, per ogni *cluster*, del prodotto fra dimensione del *cluster* e distanza fra centroide del *cluster* e centroide di tutti gli elementi.

$$BSS = \sum_{i} |C_i|(m - m_i)^2$$

- **ICC**: metrica che sintetizza in un solo valore la qualità del processo di *clustering*. Il suo valore varia fra 0 ed 1, dove 0 indica qualità pessima ed 1 indica qualità ottima. Per qualità pessima si intende la presenza di dati molto diversi fra loro all'interno dello stesso *cluster* e vicinanza fra i vari *cluster*. È pari al rapporto fra BSS e la somma di BSS e WSS.

$$ICC = \frac{BSS}{BSS + WSS}$$

Gli algoritmi di MinHash e LSH, descritti nella sezione 3.2.2.4, presentano diversi parametri da dover calibrare. Come già detto, la calibrazione di questi parametri è importante per ottenere risultati soddisfacenti nel momento in cui si applicano effettivamente i due algoritmi; inoltre, questa dipende strettamente dal tipo di dati che si sta trattando, per cui non può essere fatta su dati diversi da quelli di interesse.

In particolare, i parametri da dover calibrare sono:

- **Numero di funzioni di hash da utilizzare**, *n*
- **Dimensione degli shingles in cui dividere il documento,** ovvero numero di token consecutivi da considerare come singolo elemento, *k*
- **Dimensione delle bande in cui suddividere la matrice di MinHash**, *b*
- **Soglia di Jaccard per identificare due documenti come simili**, *j*

*** OMISSIS ***

Ultima fase del progetto è stata quella di valutare la bontà del sistema di annotazione proposto. A tal fine, il *dataset* etichettato è stato analizzato mediante due algoritmi di *machine learning*, Random Forest e C4.5, usando come lavoro di ispirazione quello di Chen et Al. [6].

In questo lavoro vengono valutate le *performance* di sei diversi algoritmi di *machine learning* (Random Forest, C4.5, Support Vector Machine, k-Nearest Neighbours, Reti Bayesiane, Bayes Naive) al variare di alcuni parametri come il rapporto fra *spam* e *non-spam* all'interno del *dataset*. Alla fine, come esposto dagli stessi autori, gli algoritmi che producono i risultati migliori sono C4.5 e Random Forest, sia in termini di *f-score* che di *true positive rate* e *false positive rate*.

Per questi motivi sono stati utilizzati i due algoritmi prima anticipati, che vengono brevemente introdotti nei prossimi paragrafi.

4.7.1 - Random Forest

Random Forest è un algoritmo di *machine learning* semplice e flessibile, che produce ottimi risultati la maggior parte delle volte. È inoltre uno dei più usati in letteratura, sia per la sua semplicità, sia perché può essere usato sia per compiti di classificazione che di regressione.

Parte fondamentale dell'algoritmo sono i *Decision Tree*, strumenti di supporto per effettuare decisioni. In breve, dato un certo numero di dati di addestramento, un *Decision Tree* produce delle regole per effettuare la classificazione. Quindi, dando in input ulteriori dati, effettua la classificazione di questi sulla base delle regole prima costruite.

Random Forest è dunque un algoritmo di *learning* supervisionato che opera come segue: a partire dai dati usati per l'addestramento, genera diversi *decision tree*, ognuno dei quali considera un sottoinsieme randomico delle *features* ed ha accesso solo ad un sottoinsieme randomico dei dati di addestramento. Questo perchè anzichè cercare la *feature* più importante, cerca la migliore fra un sottoinsieme randomico delle stesse.

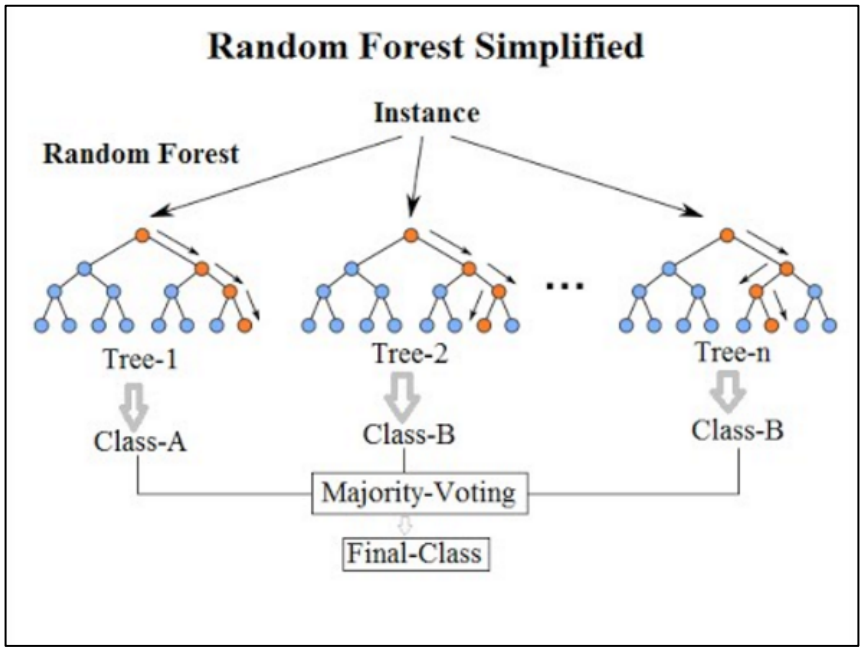

***Figura 16**: Esempio semplificato di Random Forest. In questo caso, a partire dalla singola istanza, vengono generati n Decision Tree, ognuno dei quali classifica l'istanza in un certo modo. Alla fine, viene scelta la classe presente in maggioranza fra tutti i Decision Tree generati.*

Questo porta ad una grande diversità, che generalmente conduce ad un modello ed a risultati migliori.

Al termine della fase di addestramento, i dati di testing sono dati in input ai Decision Tree, ognuno dei quali propone un'etichetta per ogni istanza. Alla fine, per ogni istanza viene effettuata una votazione di maggioranza e viene assegnata un'etichetta di conseguenza.

Un esempio semplice di Random Forest è riportato in *Figura 16*.

4.7.2 – C4.5

Anche C4.5, così come Random Forest, è un algoritmo di *learning* supervisionato basato sull'utilizzo dei *Decision Tree*.

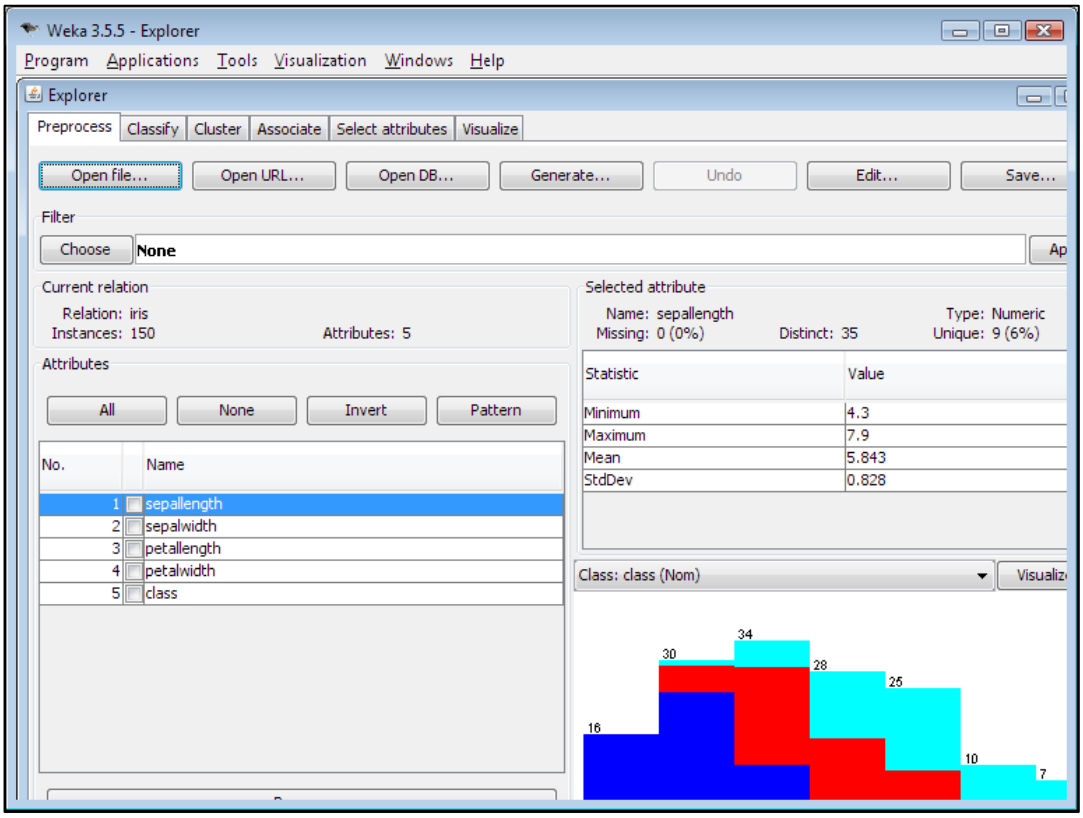

***Figura 17**: Interfaccia di WEKA Explorer ed esempio di statistiche sui dati che è in grado di fornire. Come si può vedere, per ogni diversa features produce valori come media, deviazione standard, minimo, massimo e permette di visualizzare rapidamente, sotto forma di grafico, i valori di queste features per ogni classe.*

A differenza del *Random Forest*, tuttavia, ad ogni nodo di ogni *Decision Tree* sceglie la *feature* che meglio divide l'insieme di addestramento in sottoinsiemi "arricchiti" in una classe, secondo il criterio di *information gain*, una metrica inversamente proporzionale alla probabilità che l'evento succeda. Dunque, fra tutte le *feature*, viene scelta quella col più alto *information gain* normalizzato per effettuare la decisione.

Sinteticamente, più si scende nel *Decision Tree*, minore è l'entropia e più semplice è produrre una decisione finale sui dati.

Inoltre, viene utilizzato il *Pruning*, ovvero la rimozione di *branch* non promettenti per il processo di decisione, per evitare il problema dell'*overfitting*.

4.7.3 – Implementazione

Strumento fondamentale è stato WEKA, un software progettato dall'Università di Waikato, in Nuova Zelanda, contentente una collezione di algoritmi di *machine learning* già implementati e strumenti per effettuare la classificazione dei dati. In *Figura 17* è possibile osservare l'interfaccia grafica di WEKA ed un esempio di statistiche sugli attributi che il *software* è in grado di restituire.

In breve, per come è stato utilizzato durante questo lavoro, WEKA prende in *input* un *file* formattato secondo determinate regole e salvato in formato *.arff*, contenente per ogni riga le varie *features* estratte da ogni utente e l'etichetta assegnata dal sistema di etichettatura qui proposto, ovvero la classe di appartenenza dell'utente. Quindi, permette di scegliere un algoritmo di *machine learning* ed esegue la classificazione dei dati forniti utilizzando l'algoritmo scelto. Alla fine, mostra una schermata riepilogativa dei risultati e fornisce statistiche dettagliate su questi.

*** OMISSIS ***

CAPITOLO 5: Conclusioni e Lavori Futuri

Il problema dello *spam* sui *social network* è sempre più attuale: l'esplosione della popolarità di questi ultimi ha portato alla crescita esponenziale del numero di *spammer*, utenti malevoli. Diversi casi sono diventati talmente rilevanti da attirare l'attenzione dei *media*, non ultima la manomissione dei risultati elettorali durante le elezioni presidenziali del 2016 negli Stati Uniti mediante la diffusione di false notizie e, quindi, la manipolazione dell'opinione pubblica.

Di conseguenza, i *social network* stessi stanno prendendo contromisure sempre più stringenti per combattere lo spam: Twitter, ad esempio, ha reso necessaria l'autenticazione mediante *email* o numero di cellulare e ha deciso di bloccare qualunque attività per utenti considerati possibili *spammer*.

*** OMISSIS ***

Per quanto riguarda lavori futuri e possibili estensioni, si potrebbe cercare di superare le limitazioni del sistema appena proposto, come la possibilità di analisi sui soli *tweet* in lingua inglese. L'espansione ad altre lingue comporterebbe l'implementazione di altri algoritmi di *stemming*, uno per ogni lingua che si vuole utilizzare, e l'individuazione di *spam-words* e *stop-words* anche in altre lingue.

Altro lavoro futuro potrebbe cercare di superare la principale problematica incontrata durante questo lavoro, ovvero la limitazione temporale imposta dalle API Twitter. Si potrebbe, ad esempio, progettare un sistema in cui più *client* in parallelo procedono alla raccolta dei dati.

Inoltre, applicando opportune modifiche al sistema e alle *features* da questo utilizzate nei vari passi, l'analisi potrebbe essere estesa ad altri *social network*, come Facebook. Ulteriori *features* potrebbero essere considerate per distinguere l'attività degli *spammer* da quella degli utenti genuini, in modo da migliorare le *performance* del sistema in fase di *clustering*.

Infine, i risultati dell'analisi potrebbero essere utilizzati per la realizzazione di un sistema di gestione della reputazione [21][22][23] in grado di assegnare un grado di affidabilità ai diversi utenti della rete in base ai contenuti da essi condivisi.

INDICE DELLE FIGURE

FIGURA 1 – Utenti attivi mensilmente sui Social Media 8

FIGURA 2 – Esempio di Tweet 11

FIGURA 3 – Esempio di risposta in JSON 12

FIGURA 4 – Riepilogo processo di cattura dei dati 16

FIGURA 5 – Riepilogo processo di analisi 20

FIGURA 6 – Esempio Tokenizzazione 24

FIGURA 7 – Esempio di applicazione dello Stemming 26

FIGURA 8 – Esempio di risposta di curl e get_real_link 28

FIGURA 9 – Esempio di MinHash 30

FIGURA 10 – Esempio di LSH 31

FIGURA 11 – Pseudocodice QTC 34

FIGURA 12 – Pseudocodice DrN QTC 35

FIGURA 13 – Spiegazione delle metriche utilizzate 41

FIGURA 14 – Risultati calibrazione MinHash, LSH 45

FIGURA 15 – Risultati calibrazione NDC 48

FIGURA 16 – Esempio di Random Forest 55

FIGURA 17 –Interfaccia di WEKA 56

INDICE DELLE TABELLE

TABELLA 1 – Endpoint delle API Twitter 10

TABELLA 2 – Esempio di keywords “spammy” 18

TABELLA3 – Esempio di stop-words in inglese 25

TABELLA 4 – Esempio di regole di sostituzione dello stemming di Porter 26

TABELLA 5 – Servizi di URL Shortening più comuni 28

TABELLA 6 – Test sull’efficienza delle varianti del QT 36

TABELLA7 – Esempio di spam-words 38

TABELLA 8 – Esempio features NDC 47

TABELLA 9 – Risultati calibrazione QTC 51

TABELLA 10 – Risultati testing manuale del dataset 53

TABELLA 11 – Riepilogo risultati WEKA 58

BIBLIOGRAFIA

[1] A. MITCHELL, J. KILEY, J. GOTTFRIED, and E. GUSKIN. The role of news on facebook. October 2013.

[2] F. Concone, P. Ferraro, G. Lo Re. Towards a Smart Campus through Participatory Sensing. In Proceedings of the 4th IEEE International Conference on Smart Computing (SMARTCOMP 2018)

[3] F. Concone, S. Gaglio, G. Lo Re, M. Morana. Smartphone Data Analysis for Human Activity Recognition. In Proceedings of the 16th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2017)

[4] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda. All your contacts are belong to us: automated identity theft attacks on social networks. In Proceedings of the 18th international conference on World wide web, WWW ’09, pages 551–560, New York, NY, USA, 2009. ACM.

[5] A. De Paola, S. Gaglio, G. Lo Re and M. Morana. A Hybrid System for Malware Detection on Big Data. In Proceedings of the IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)

[6] Chen, Chao, et al. "6 million spam tweets: A large ground truth for timely Twitter spam detection." Communications (ICC), 2015 IEEE International Conference on. IEEE, 2015.

[7] F. Benevenuto, T. Rodrigues, V. Almeida, J. Almeida, and M. Gonçalves. Detecting spammers and content promoters in online video social networks. In Int’l ACM Conference on Research and Development in Information Retrieval (SIGIR), 2009.

[8] S. Garriss, M. Kaminsky, M. Freedman, B. Karp, D. Mazi`eres, and H. Yu. Re: Reliable email. In USENIX Conference on Networked Systems Design & Implementation (NSDI), 2006.

[9] Sedhai, Surendra, and Aixin Sun. "Hspam14: A collection of 14 million tweets for hashtag-oriented spam research." Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, 2015.

[10] C. Grier, K. Thomas, V. Paxson, and M. Zhang. @spam: the underground on 140 characters or less. In Proceedings of the 17th ACM conference on Computer and communications security, CCS '10, pages 27–37, New York, NY, USA, 2010. ACM.

[11] K. Lee, J. Caverlee, and S. Webb. Uncovering social spammers: social honeypots + machine learning. In Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval, SIGIR '10, pages 435–442, New York, NY, USA, 2010. ACM.

[12] J. Song, S. Lee, and J. Kim. Spam filtering in twitter using sender-receiver relationship. In Proceedings of the 14th international conference on Recent Advances in Intrusion Detection, RAID'11, pages 301–317, Berlin, Heidelberg, 2011. Springer-Verlag.

[13] Egele, Manuel, et al. "Towards detecting compromised accounts on social networks." IEEE Transactions on Dependable and Secure Computing 1 (2017): 1-1.

[14] F. Concone, A. De Paola, G. Lo Re, M. Morana. Twitter Analysis for Real-Time Malware Discovery. In Proceedings of the International Annual Conference of AEIT (2017)

[15] S. Gaglio, G. Lo Re, M. Morana. A framework for real-time Twitter data analysis. In Journal of Computer Communications, Elsevier, ISSN 0140-3664

[16] S. Gaglio, G. Lo Re, M. Morana. Real-Time Detection of Twitter Social Events from the User's Perspective. In Proceedings of the 2015 IEEE International Conference on Communications (ICC2015)

[17] Danalis, Anthony, Collin McCurdy, and Jeffrey S. Vetter. "Efficient quality threshold clustering for parallel architectures." Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International. IEEE, 2012.

[18] Heyer, Laurie J., Semyon Kruglyak, and Shibu Yooseph. "Exploring expression data: identification and analysis of coexpressed genes." Genome research 9.11 (1999): 1106-1115.

[19] Loforte Jr, Frank. "Efficient Variations of the Quality Threshold Clustering Algorithm." (2015).

[20] Tao, Ke, et al. "Groundhog day: near-duplicate detection on twitter." Proceedings of the 22nd international conference on World Wide Web. ACM, 2013.

[21] V. Agate, A. De Paola, G. Lo Re, M. Morana. A Simulation Framework for Evaluating Distributed Reputation Management Systems. In Proceedings of the 13th International Conference on Distributed Computing and Artificial Intelligence

[22] V. Agate, A. De Paola, S. Gaglio, G. Lo Re, M. Morana. A Framework for Parallel Assessment of Reputation Management Systems. In Proceedings of the 17th International Conference on Computer Systems and Technologies. CompSysTech 16

[23] C. Crapanzano, F. Milazzo, A. De Paola, G. Lo Re. Reputation management for distributed service-oriented architectures. In Proceedings of the Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop (SASOW), 2010, pp. 160-165

SITOGRAFIA

[24] https://www.nytimes.com/2017/09/07/us/politics/russia-facebook-twitter-election.html

[25] https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/

[26] https://www.bloomberg.com/news/articles/2018-06-26/twitter-ramps-up-fight-against-abuse-and-malicious-bots

[27] https://techcrunch.com/2018/06/21/twitter-acquires-anti-abuse-technology-provider-smyte

[28] https://gist.github.com/kimchy/1626305

[29] http://www.andreaminini.com/ir/stemming/

[30] https://www.statista.com/statistics/266162/url-shortener-phishing-usage/

[31] https://www.lifewire.com/shortening-long-links-3486603