Università degli Studi di Bari - "Aldo Moro"

FACOLTÀ DI SCIENZE MM.FF.NN.
Corso di Laurea Magistrale in Informatica

Tesi di laurea
in
Basi di conoscenza e Data Mining

TITOLO TESI

Relatori:
Chiar.mo Prof. Michelangelo Ceci
Chiar.mo Prof. Donato Malerba

Correlatori:
Dott.ssa Fabiana Lanotte
Dott. Roberto Corizzo

Laureando:
Rocco Caruso
Matricola 573394

Anno Accademico 2014-2015

Ahi mè ...

Indice

Elenco delle tabelle

Elenco delle figure

Abstract

Twitter è ad oggi il servizio di micro-blogging più utilizzato in assoluto, con circa 284 milioni di utenti attivi al mese, ogni giorno vengono prodotti oltre 500 milioni di tweets. Gli utenti di Twitter, possono pubblicare degli status, o *tweets*, non più lunghi di 140 caratteri. Se questo vincolo da una parte costituisce un forte limite, dall'altro rappresenta una delle caratteristiche fondamentali di Twitter: *l'immediatezza*. Questo limite costringe gli utenti ad produrre messaggi molto sintetizzati, quasi come slogan, che quindi sono più facili da diffondere. Grazie a queste caratteristiche, qualsiasi persona che assiste o è coinvolta in un *evento*, è in grado di diffondere informazioni in *real-time*. Talvolta, i tweets (Twitter microblog posts) diffondono notizie anche più velocemente dei media tradizionali (come la morte di Micheal Jackson [1]). Bisogna però sottolineare che questi tweet che possono riflettere eventi, rappresentano solo una piccola percentuale di tutti i tweet prodotti. La maggior parte infatti, è costituita da status personali, messaggi anche privi di senso, spam. Risulta quindi necessario un sistema che sia capace di scoprire "eventi"o "topics"da questo flusso di dati. Affinché questi eventi abbiano una qualche utilità, è altresì necessario, che siano rilevati con una bassa latenza. I classici algoritmi di Data Mining, non riescono a scalare all'aumentare della mole dei dati . Questo lavoro di tesi ha l'obiettivo di applicare algoritmi di data Mining per la scoperta di eventi, attraverso un'architettura di calcolo distribuita: Apache SPARK.

[1] ttp://www.dailymail.co.uk/sciencetech/article-1195651/How-Michael-Jacksons-death-shut- Twitter-overwhelmed-Google–killed-Jeff-Goldblum.html

Capitolo 1

Introduzione

Scoprire nuovi eventi o topics da Twitter, non è affatto un task banale sia sia la mole dei dati (oltre 400 milioni di tweet giornalmente), che per la natura stessa dei tweets. Se da un lato il limite a 140 caratteri ne rende più semplice la diffusione on-line, dall'altro complica ulteriormente il task, poiché gli utenti spesso,proprio a causa di tale limite,ricorrono a slang, vocaboli OOV[1] o emoticons.

1.1 Twitter

Twitter è ad oggi il servizio di "microblogging" più diffuso e con il più alto tasso di crescita. Negli anni la sua popolarità crescente ha anche attirato anche un alto numero di ricercatori, come si può notare dall'alto numero di articoli riguardanti Twitter che sono stati pubblicati in numerosi campi di ricerca. Sebbene il termine "microblog" spesso possa indurre intendere tale servizio come una versione "micro" di un blog, sono due due media molto diversi fra loro [8]. I blog infatti sono progettati, principalmente per permettere ad utenti, di fornire commenti e opinioni su topic di cui sono esperti, gli autori devono anche garantire una certa validità dei contenuti.

[1]Out Of Vocabolary

Dall'altra parte, i microblog, come Twitter, sono invece pensati per permettere di condividere opinioni, news, ma in maniera molto concisa (max 140 caratteri) proprio per far sì che si abbia una diffusione tempestiva delle informazioni. Proprio grazie a questa caratteristica, i tweets possono essere pubblicati mediante dispositivi mobili, consentendo a chiunque sia testimone di un qualsiasi evento di diffondere la notizia in real-time. Twitter inoltre è anche un servizio di social networking, ogni utente può ricevere gli aggiornamenti ("follow") di altri utenti senza previa approvazione. Questa relazione è asimmetrica e può essere concettualizzata come una Directed social network o *follower network*

1.1.1 Twitter come fonte di informazione

Molte notizie sono state diffuse su Twitter anche prima della diffusione sui media classici. Uno degli esempi più significati è stato rappresentato dalla notizia della di Michael Jackson del 2009. Alle 2:26pm del 24 Giungo 2009, la notizia trapelò su Twitter e fu diffusa in una maniera così virale che che Google la identificò come un attacco hacker.t . La validità della notizia fù verificata da Google solo 25 minuti dopo, solo allora i media mainstream iniziarono a far diffondere la notizia [2].Anche nel caso del terremoto in Abruzzo del 6 aprile 2009, gli utenti Twitter hanno segnalato la notizia prima dei media tradizionali.

1.2 Big Data

Sebbene oggigiorno il termine "big data"sia molto in voga, la sua definizione è ancora piuttosto vaga. Alcuni definizioni fanno riferimento al volume dei dati, altre invece fanno riferimento alla ricchezza dei dati. Per altri, i "big data"sono quei dati troppo grandi per gli standard tradizionali, ovvero quando il volume dei dati supera petabytes o zettabytes. Altri ancora intendono

[2]ttp://www.dailymail.co.uk/sciencetech/article-1195651/How-Michael-Jacksons-death-shut- Twitter-overwhelmed-Google–killed-Jeff-Goldblum.html

per big data, quei dati che riescono ad esprimere più sfaccettature delle stesse entità che rappresentano, che se fossero memorizzati nei classici database relazionali (RDBMS) avrebbero migliaia di colonne. L'aggettivo "big"non si riferisce soltanto al volume dei dati, ma anche alla loro complessità. Esistono infatti, molti piccoli datasets che sono considerati big data che non richiedono molo spazio di memorizzazione, ma hanno una complessità intrinseca molto alta. Allo stesso tempo, datasets che richiedono molto spazio di memorizzazione possono non essere abbastanza complessi, da essere considerati Big data. Il solo volume dei dati, quindi non basta a definire questi big-data. Una definizione molto diffusa è quella delle tre *V*, dove, oltre al volume dei dati, si considera anche la Velocità e la Varietà . Per Velocità si intende che i dati sono generati con un'elevata frequenza. La Varietà si riferisce al fatto che i dati possono essere strutturati, semi strutturati o non strutturati affatto : dati transazionali, video, audio testo file di log. In aggiunta a queste tre V, talvolta viene aggiunta una quarta alla definizione: *Veridicità*. La veridicità è un indicazione dell'integrità di questi dati e si riferisce al livello di trust in questi dati stessi, affinché si possano utilizzare nei processi decisionali Analizzare questi big-data può permettere di prendere decisioni in maniera più oculata e molto più velocemente, usando dati che in passato erano inaccessibili o inutilizzabili. Per tali ragioni, i classici database relazionali non riescono a gestire facilmente questi dati.

1.3 Hadoop

Hadoop è stato uno dei primi sistemi open-source più popolari per processare i big data. É un sistema scalabile e fault-tolerant che consente di processare grandi dataset attraverso un cluster di server.

Uno dei fattori di successo di Hadoop è il basso costo. Hadoop è un sistema open-source che può essere eseguito su un cluster di commodity[3] hardware. Riesce cioè a garantire facilmente una scalabilità orizzontale aggiungendo

[3]hardware non dedicato di basso costo

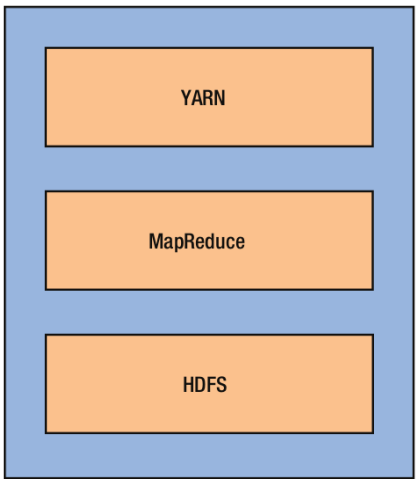


Figura 1.1: componenti principali di Hadooop

al cluster, dei server poco costosi. Inoltre, riesce a garantire, via software la tolleranza ai guasti: assume che prima o poi vi saranno dei guasti e li gestisce in maniera trasparente. Uno sviluppatore software non dovrà preoccuparsi di gestire questi fault. Consente, cioè, di sviluppare applicazioni distribuite in maniera molto più semplice poiché viene sperata la logica di elaborazione dei dati dalla logica di distribuzione dei dati. Hadoop è composto da tre componenti principali: un cluster mananger (YARN), un modello di computazione distribuito (Map-reduce) e un file system distribuito (HDFS)

1.3.1 Map Reduce

MapReduce è il modello di computazione distribuita fornito da Hadoop. Mentre HDFS fornisce un file system distribuito per memorizzare grandi dataset, MapReduce fornisce un framework di computazione che consente di processare grandi dataset in parallelo attraverso un cluster di computer (nodi). Questo modello astrae la computazione distribuita fornendo dei costrutti ad alto livello che consentono di sviluppare facilmente applicazioni distribuite. Il framework MapReduce schedula in maniera automatica l'esecuzione di un applicazione su un insieme di macchine in un cluster, prendendosi carico della gestione della comunicazione fra nodi, del bilanciamento

del carico di esecuzione fra i nodi e dei possibili guasti dei nodi. In questo modo, gli sviluppatori possono concentrarsi sulla logica di processing dei dati tralasciando questi dettagli.

Come suggerisce il nome stesso, questo modello si basa su due funzioni :*map* e *reduce*. Tutti i carichi di lavoro in un appplicazione MapReduce sono espressi implementando queste due funzioni. La funzione *map* riceve in input una coppia chiave-valore e restituisce a sua volta una insieme di coppie chiave valore intermedie. Il framework MapReduce esegue la funzione map per ogni coppia chiave valore presente nel dataset di input. L'output delle funzioni Map, è ordinato e raggruppato in base ai valori di chiave intermedi, e costituirà l'input della funzione Reduce. La funzione *Reduce* aggrega i risultati della funzione map, in base al valore di chiave intermedio.

I dati possono essere sia semi strutturati o non strutturati affatto, non è richiesto che i dati siano conformi ad uno schema rigido predefinito. L'unico requisito è che sia possibile esprimere il dataset in input come una serie di coppie chiave valore. Il modello MapReduce ha rivoluzionato il modo di processare grandi datasets, offrendo un modello semplice che consente di scrivere programmi che possono essere eseguiti in parallelo su molte macchine. Grazie a questo, Map-Reduce consente di ottenere una scalabilità orizzontale: all'aumentare della dimensione dei dati è possibile aggiungere nuove macchine mantenendo quasi invariato il tempo di esecuzione.

1.4 Apache Spark

Apache Spark [**?**] è una framework open-source che consente di processare grandi dataset in maniera distribuita. Spark può essere considerato come il successore del modello MapReduce di Hadoop, entrambi i framwork sono progettati per poter processare i big data. Spark mantiene inalterata la scalabilità di MapReduce e la tolleranza ai guasti, e inoltre aggiunge nuove caratteristiche, offrendo inoltre ulteriori vantaggi rispetto a MapReduce. Primo fra questi è la velocità: Spark riesce a processare i dati riducendo di molto

i tempi di latenza (fino a 100 volte), poiché consente ai nodi di processing di immagazzinare in memoria centrale i risultati intermedi, a differenza di MapReduce, dove invece erano serializzati sul file system. La velocità può essere talvolta un fattore determinante. Impiegare troppo tempo per processare i dati, rallenta tutto il processo decisionale riducendo il valore stesso dei dati. La principale astrazione fornita da SPARK sono i **R**esilent **D**istributed **D**ataset (RDD) [**?**], che essenzialmente sono una collezione immutabile e distribuita di oggetti. Questi oggetti sono, suddivisi in più partizioni che generalmente sono distribuite su più nodi. Questa astrazione permette agli sviluppatori di materializzare i risultati intermedi della computazione nella memoria dei vari nodi di processing. Ciò signfica che i prossimi step che vogliono riconsultare questi dati, non li dovranno rielaborare o ricaricare da disco. Per tale ragione, Spark si presta bene per eseguire in parallelo sia algoritmi altamente iterativi che richiedono di scansionare un dataset di input più volte, come gli algoritmi di machine-leanring.

Questi RDD possono essere creati in due modi: o a partire da dati in un sistema di memorizzazione stabile[4] (HDFS, Hive, Cassandra,) o a partire da altri RDD. Queste operazioni, che creano RDD, sono dette *trasformazioni*.

Spark non materializzare gli RDD dopo ogni operazione, verrà memorizzato per ogni RDD intermedio il suo *lineage* ovvero la sequenza di trasformazioni che lo ha prodotto, a partire da un altro RDD. In tal modo SPARK nel caso vi sia un failure, può rielaborare un RDD in maniera del tutto trasparente.

Architetturalmente, Spark è progettato per essere altamente accessibile: offre API per Python, Java, Scala, SQL e R. Inoltre si integra alla perfezione con strumenti di Big Data quali Hadoop (e strumenti che dipendono da esso come HBase, Hive, etc.) e Cassandra.

La figura 1.2 mostra le principali componenti di Apache Spark:

- **Spark Core**: contiene le funzionalità base di Spark fra cui l'astrazione sulla quale si basa l'intero ambiente, i *resilient distributed dataset* (RDD).

[4]per stabile si intende fault-tolerant

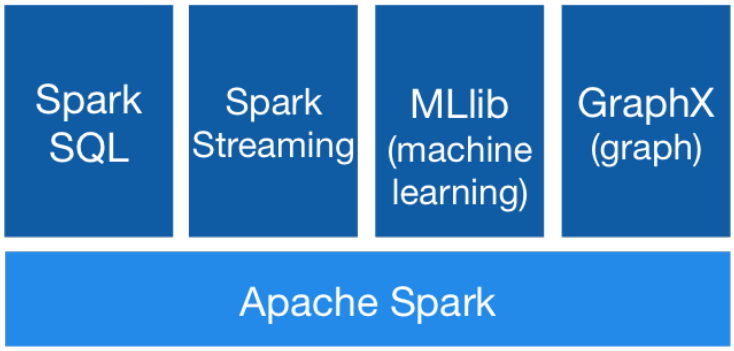


Figura 1.2: Stack applicativo Apache Spark

- **Spark SQL**: contiene le componenti per elaborare e manipolare dati strutturati. Permette l'interrogazione (tramite SQL) di dabatase relazionali, JSON, HIVE (mediante lo Hive Query Language) e file Parquet.
- **Spark Streaming**: componente che permette l'elaborazione di stream di dati.
- **MLLib**: contiene algoritmi di Machine Learning per vari tipi di task quali classificazione, regressione, clustering, collaborative filtering, frequent pattern mining, dimensionality reduction, feature extraction e statistica di base.
- **GraphX**: libreria per la manipolazione ed estrazione di conoscenza in grafi.

1.4.1 Modello di esecuzione

Una applicazione Spark consiste in un processo *driver* e di un insieme di processi *executors* distribuiti sui nodi del cluster.

Fra le responsabilità del driver si annoverano:

- interazione con l'utente;
- coordinare e distribuire il flusso di controllo.

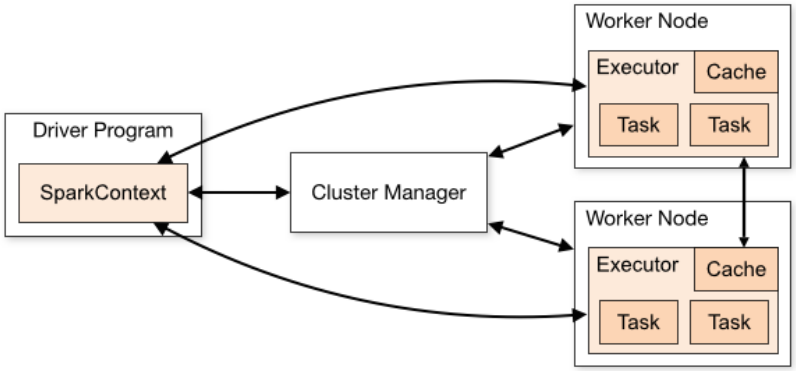


Figura 1.3: componenti principali di Hadooop

I processi "executor" sono responsabili dell'esecuzione dei work in forma di *tasks* e di memorizzare qualsiasi tipo di dato che l'utente sceglie di mantenere in cache.

In cima al modello di esecuzione vi sono i *jobs*: l'invocazione di una azione all'interno di una applicazione Spark provoca l'esecuzione di uno Spark job in grado di soddisfare la richiesta. Spark esamina il grafo degli RDD dal quale l'zione dipende e produce un piano di esecuzione che inizia computando le risorse (gli RDD) in ordine di dipendenza, culminando con l'RDD che produrrà i risultati dell'azione.

Il piano di esecuzione consiste nell'assemblare trasformazioni di jobs in *stages*. Uno stage corrisponde a una collezione di *tasks* che eseguono simultaneamente lo stesso codice su una differenze partizione dei dati: ciascuno stage contiene una sequenza di trasformazioni che possono essere completati senza rimescolare i dati.

Quest'ultimo aspetto inficia fortemente le performance dell'applicazione. Una partizione di dati subisce un processo denominato "shuffling" quando vengono effettuate trasformazioni *narrow*, ad esempio una *map*.

Il trasferimento dei dati fra i nodi del cluster avviene mediante un meccanismo di serializzazione: dopo questo processo, i dati vengono memorizzati in una cache e trasferiti in rete per uno shuffling. Spark prevede due meccanismi di serializzazione: l'approccio classico (utilizzando le API Java disponi-

bile a partire dall'interfaccia *java.io.Serializable*) e mediante la libreria *Kryo*. Quest'ultima prevede un formato molto più compatto in grado di svolgere le operazioni in maniera più rapida.

Capitolo 2

Stato dell'Arte

2.1 Event detection nei media tradizionali

L'attività di event-detection è stata a lungo utilizzata per individuare eventi da stream testuali derivanti dai media più tradizionali come giornali o radio. infatti l'event-detection è stata per molto tempo oggetto di ricerca del programma di *Topic Detection and Tracking TDT* [2], un' iniziativa promossa dalla DARPA [1], con lo scopo di organizzare stream di notizie testuali sulla base degli eventi di cui discutono. Secondo il TDT, l'obiettivo dell'attività di *event detection*, è scoprire nuovi eventi o eventi precedentemente non noti, a partire da stream di notizie testuali derivanti dai media tradizionali come notiziari o newswire, dove ciascun evento è definito come segue:
definizione 1 (Evento)**.** *: qualcosa, non banale, che accade in un luogo e tempo specifico*

Le tecniche di event-detection possono essere classificate in due macro categorie: *document-piovot* e *document piovot* a seconda che utilizzino feature dei documenti o feature temporali delle singole keywords presenti nei documenti. La prima scopre eventi effettuando un clustering dei documenti sulla base di una qualche funzione di distanza fra i documenti stessi [24], men-

[1] Agenzia di ricerca agenzia per i progetti di ricerca avanzata per la difesa

tre nella seconda si studia la distribuzione delle singole parole e scoprono nuovi eventi raggruppando le parole [13] Come evidenziato da [24] infatti, l'event detection può essere ricondotto al problema della scoperta di pattern in uno stream testuale, quindi il modo più naturale per scoprire nuovi eventi, è quello di usare un algoritmo di clustering. Il task di event-detection si può suddividere in tre fasi principali: data preprocessing, data rappresentation, data organization o clustering. Nella fase di preprocessing vengono applicate al testo delle classiche tecniche di NLP come la rimozione di stopwords, tokenizzazione e stemming. I modelli di rappresentazione di dati più utilizzati per l'event detection sono *il modello vettoriale* e *il modello bag of words*, i cui elementi saranno diversi da zero, se il termine corrispondente è presente nel documento. A ciascun termine nel vettore, è assegnato un peso secondo lo schema *tf-idf* [20] che valuta quanto è importante una parola per un documento all'interno di un corpus. Questo modello di rappresentazione non prende in considerazione l'ordine temporale delle parole ne le caratteristiche sintattiche o semantiche del testo come il part of speech tag o named entities. Per questa ragione utilizzando questo modello, ad esempio, sarebbe difficile distinguere due eventi simili ma accaduti ad un mese di distanza fra loro. Nel lavoro di [24] il task di scoperta di nuovi eventi da uno stream testuale di news è suddiviso in due fasi principali: Retrospective Event Detection (RED), New Event Detection (NED). La prima fase (RED) comporta la scoperta di eventi da una collezione già nota di documenti, mentre nella seconda si cerca di identificare gli eventi dallo stream di notizie in tempo reale. Per il RED è stato utilizzato un algoritmo di clustering gerarchico: *Group Average Clustering GAC*, che consente anche di descrivere gli eventi identificati con diversi livelli di granularità. Per la fase di New Event Detection, invece, solitamente viene adottato un agoritmo di clustering incrementale single-pass [2, 24] che consente di suddividere i documenti nei vari cluster non appena arrivano dallo stream. In particolare ciascun documento viene elaborato sequenzialmente e viene assorbito dal cluster più simile, o verrà creato un nuovo cluster se la similarità è al di sotto di una soglia prestabilita. In un ambiente di detection on-line (NED) un forte vincolo è costituito dal fatto che non si può hanno informazioni di eventi futuri,ovvero non è possibile utilizzare dati provenien-

ti da documenti successivi, cronologicamente, a quello corrente. Utilizzando un modello di rappresentazione vettoriale, questo vincolo pone delle problematiche su come gestire la crescita del vocabolario dei termini quando vengono aggiunti nuovi documenti al corpus e come modificare delle statistiche inerenti l'intero corpus come l'IDF. La soluzione suggerita da [24] è quella di modificare il vocabolario dei termini in maniera incrementale e modificare l'IDF ogni qual volta viene aggiunto un nuovo documento.

$$idf_t(w) = log_2 \left(\frac{N_t}{df_t(w)} \right) \tag{2.1}$$

dove N_t è il numero di documenti fino al tempo t e $df_t(w)$ è la document frequency della keyword w fino al tempo t. In pratica questi approcci NED, tendono a divenire molto costosi sia in termini di risorse computazionali che di tempo richiesto, e in taluni casi addirittura irrealizzabili se non utilizzando delle tecniche che ne migliorino l'efficienza. Una possibile tecnica per ridurre i costi è quella di utilizzare una *time window* [15, 17] per limitare il numero vecchi documenti da analizzare quando si prende in considerazione un nuovo documento. Utilizzare una finestra temporale non solo riduce i costi, ma permette anche di limitare lo scope degli eventi scoperti, consentendo di identificare eventi simili ma che accadono in uno slot temporale diverso [24]. Tutte queste tecniche per il TDT si basano sull'assunzione che tutti i documenti siano rilevanti e contengono informazioni di eventi, poichè lavorano su stream di informazioni affidabili, assunzione che è chiaramente violata per quanto riguarda lo stream di Twitter. Nelle tecniche feature-pivot un evento viene invece modellato come una attività che presenta picchi di frequenza (burst), ovvero un evento è rappresentato dall'insieme di keywords che presentano un burst [2]. L'assunzione fatta da queste tecniche è che alcune parole avranno un incremento di utilizzo repentino quando accade un evento. Nel lavoro di [13] viene utilizzato un'automa a stati infiniti per poter identificare i burst delle keyword all'interno dello stream testuale. Gli stati dell'automa corrispondono alla frequenze delle singole parole, mentre le transizioni fra a gli stati identificano i burst che corrisopondo a un cambiamento significativo nella frequenza. A differenza delle tecniche document-pivot, in

questo caso si cercano di identificare eventi raggruppando (ovvero effettuando il clustering) quelle keyword che presentano un burst, piuttosto che i documenti. Nel lavoro di [2] la frequenza delle parole viene modellata tramite una distribuzione bionomiale, poi vengono individuate le bursty-keywords sulla base di una soglia euristica, per poi raggrupparle al fine di identificare gli eventi.

2.2 Event Detection in Twitter

L'attività di Event Detection nei microblogs come Twitter, è concettualmente molto simile all' Event Detection nei media tradizionali. In entrambi i casi, viene dato in input al sistema uno stream di documenti testuali e l'obiettivo è quello di scoprire degli eventi raggruppando i documenti o le singole parole contenute nei documenti stessi. L'unica differenza è data tipo e il volume di documenti dello stream che devono analizzare, in pratica tuttavia, questa unica differenza si riflette in una serie di nuove sfide per il task dell'event detection. Innanzitutto il volume di documenti nel caso dei microblogs come twitter è di diversi ordini di grandezza più grande rispetto ai media tradizionali, ma soprattutto, nel caso di stream derivanti dai media tradizionali, tutti i documenti hanno una qualche rilevanza rispetto ad un avvenimento, una notizia. Nel caso dei tweet, invece, vi possono essere grandi quantità di messaggi privi di significato (pointless babbles) [11] e rumors [5]. Inoltre le caratteristiche di Twitter e la sua poplarità sono molto allettanti per spammers e altri e altri content polluters [14] per disseminare pubblicità, virus, pornografia pishing o anche per compromettere la reputazione del sistema. La sfida più grande che bisogna affrontare nell'attività di event detection per i tweet, è quindi quella di poter separare informazioni mondane e inquinate da informazioni su eventi reali. Altre difficoltà sono causate principalmente dalla brevità dei messaggi (max 140 caratteri), dall'uso di abbreviazioni, errori di spelling e grammaticali, e l'uso improprio della struttura delle frasi e l'utlizzo di più lingue nel medesimo tweet. Per queste ragioni anche le tecniche tradizionali di natural language processing meno appropriate per i

tweet. Un altro lavoro interessante che si colloca in quest'area è TwitterStand [21]: un sistema per scoprire le ultime notizie da twitter. Per poter distinguere il rumore dalle news, hanno selezionato manualmente 2000 utenti come "Seeders" ovvero utenti che pubblicano su Twitter news come stazioni televisive, giornali, bloggers etc. I tweets non appartenenti a questi seeders, invece, vengono filtrati per mezzo di un classificatore Naive Bayes. Dopo aver filtrato i tweet, viene applicato un algoritmo di clustering incrementale al fine di creare cluster tali che ognuno corrisponda ad una "news". Il modello di rappresentazione utilizzato è quello vettoriale con pesatura tf-idf e la funzione di similarità adottata è quella del coseno. Inoltre viene mantenuta una lista di cluster "attivi" per ridurre il numero di confronti da effettuare. In particolare un cluster viene definito inattivo se la media delle date di pubblicazione dei tweet, non supera i tre giorni. Una volta identificati i topic (news), il sistema cerca di localizzare ciascun cluster, ovvero cerca di assegnare una posizione geografica sia sulla base del contenuto testuale che sui geotag presenti nei tweet. Un altro sistema per scoprire news da twitter è stato proposto Phuvipadawat e Murata [19]. In questo lavoro per ridurre il rumore, i tweet vengono innanzitutto campionati utilizzando attraverso le streaming API di twitter e fornendo delle specifiche keyword da monitorare (#breakingNews, #breaking #news). I tweet raccolti vengono sucessivamente indicizzati tramite Apache Lucene[2]. I tweet simili fra loro vengono raggruppati per poter identificare news. Anche in questo caso la similarità adottata si basa sulla similarità del coseno fra le rappresentazioni tf-idf dei tweet, ma viene assegnato un *boost* per quei termini che corrispondono a nomi propri e per hashtag e username. I nomi propri sono identificati utilizzando lo Stanford Name Entity Recognizer [3] addestrato su un corpora di news tradizionali. I nuovi messaggi saranno inclusi in un cluster se sono simili al primo tweet e ai top-k termini presenti nel cluster. I cluster prodotti vengono poi ordinati sulla base dell'affidabilità (numero di followers) e popolarità (numero di retweet). Gli autori hanno fortemente sottolienato l'importanza dell'identificazione dei nomi propri al fine di migliorare il calcolo

[2]https://lucene.apache.org/core/
[3]http://nlp.stanford.edu/software/CRF-NER.shtml

della similarità fra i tweet, e di conseguenza migliorare l'accuratezza generale del sistema. Nel lavoro di [3] viene posta maggiore attenzione sull'identificazione di eventi reali da Twitter. Il metodo da loro proposto utilizza un algoritmo di clustering incrementale, che raggruppa i tweet simili fra loro e poi classifica i risultanti cluster in eventi real-world o non events. L'algoritmo di clustering utilizzato è il classico algoritmo di incrementale basato su soglia, ogni tweet è rappresentato mediante un boosted tf-idf vector, e viene utilizzata la similarità del coseno per valutare la distanza fra un tweet e il centroide di ogni cluster. Oltre ai classici step di pre-processing come tokenization, stop-word removal e stemming, viene raddoppiato il peso per gli hashtag, poiché sono considerati fortemente indicativi del contenuto del messaggio. Gli autori hanno definito quattro tipologie di feature per i cluster individuati, per poter distinguere eventi reali da non-event o "twitter center topic" :

1. temporal-features: sono state definite un insieme di caratteristiche temporali per poter caratterizzare il volume dei termini più frequenti all'interno di un cluster.

2. social-features: insieme di feature che caratterizzano il grado di interazione degli utenti nei tweet del cluster come la percentuale di retweet, di replies. L'assunzione fatta è che i cluster contenenti un alta percentuale di retweet potrebbero non contenere informazioni di eventi reali.

3. topical-features: descrivono la coerenza del cluster rispetto ad un topic. L'idea sottostante è che i cluster relativi ad eventi tendono a svilupparsi attorno ad un tema comune, al contrario di non-event cluster che si sviluppano attorno a diversi termini comuni e.g: ("sleep" or "work"). Per stimare questa coerenza, gli autori calcolano la media della similarità dei tweet rispetto al centroide.

4. twitter centric-features: queste caratteristiche hanno lo scopo di identificare le attività twitter-centric. Esempi di queste feature sono la percentuale di tweet contenenti hashtag e la percentuale di tweet contenen-

ti l'hashtag più utilizzato. Gli autori presumono che un'alta percentuale della prima stia ad indicare un topic conversazionale.

Poichè i cluster evolvono nel tempo, queste feature sono periodicamente aggiornate. Sulla base di queste feature, è stato addestrato una support vector machine (SVM) a partire da un insieme di cluster etichettati, che verrà usata per decidere se un nuovo cluster contiene o meno informazioni relative a eventi reali.

Uno dei maggiori problemi da affrontare, quando si cerca di effettuare il clustering dei documenti è il numero di confronti da effettuare, per ridurre tale numero Aggrawal e Subbian [1] utilizzano, un numero prefissato di cluster e per ciascun cluster prodotto mantengono delle informazioni che ne sintetizzano il contenuto (summaries). Ogni cluster-summary contiene:

- un node-summary che è dato dall'insieme degli utenti e le loro rispettive frequenze all'interno del clusters
- un content-summary dato dall'insieme delle keywords e i rispettivi pesi TF-IDF.

Sulla base di queste due informazioni aggregate, hanno definito una funzione di similarità che quindi non tiene solo conto del testo dei tweet, ma che sfrutti informazioni della struttura sociale di Twitter. Tale funzione di similarità tra un documento D ed un cluster C è definita come segue:

$$Sim(D, C) = \lambda \cdot SimS(D, C) + (1 - \lambda) \cdot SimC(D, C) \tag{2.2}$$

Dove $SimS$ e $SimC$ rappresentano rispettivamente la similarità strutturale e la similarità basata su contenuto, e λ rappresenta un fattore di bilanciamento compreso fra $[0, 1]$. Ogni nuovo documento proveniente dallo stream, viene assegnato al cluster più vicino a meno che la sua similarità verso ciascun cluster esistente sia inferiore in maniera *significativa* rispetto agli altri documenti. Un valore di similarità si considera come significativamente più piccolo, se è minore di $\mu - 3 \cdot \sigma$, dove μ rappresenta la media di tutti i precedenti score di similarità, e σ la deviazione standard.

AL fine di permettere di eseguire il task di scoperta di eventi su larga scala invece, in [18] è stato ideato un algoritmo che fa ricorso al *Local Sensitive Hashing*[12] grazie a cui è possibile superare alcuni limiti degli approcci classici. L'utilizzo dell'LSH, permette di ridurre drasticamente il tempo richiesto per la scoperta del Nearest Neighbor di un punto in uno spazio vettoriale. In particolare per ogni documento proveniente dallo stream, viene calcolata la distanza rispetto al suo nearest neighbor, che verrà adoperata sia per determinare il grado di novità del documento rispetto ai precedenti (Novelty Score), sia per dall'altro è utilizzato per creare dei "thread di tweets" o in altri termini cluster, che come negli altri metodi document-pivot precedentemente descritti, si assume che corrisponderanno ad eventi. Viene definita una relazione *links* come segue: si dice che un tweet a ha un *link* verso il tweet b se la loro distanza è al di sotto di uuna soglia.

$$a \; links \; b \iff 1 - cos(a, b) < t \tag{2.3}$$

Dopodiché ciascun tweet a verrà assegnato ad un thread esistente se la sua distanza dal suo nearest neighbor è al di sotto la soglia, o sarà considerato come il primo tweet di un nuovo thread. Nel primo caso il tweet a sarà assegnato al medesimo thread cui appartiene il suo nearest neighbor. Cambiando la soglia t è possibile controllare la granularità dei thread prodotti. Se t è un valore molto alto, si avranno pochi thread ma molto grandi, mentre un valore di t molto basso genererà molti thread di piccole dimensioni. Una volta individuati i thread, vengono considerati solo quelli che crescono più velocemente. Un alto fattore di crescita da infatti indicazione del fatto che la notizia del nuovo evento si stia diffondendo, per tale ragione vengono restituiti solo i threads con il grow-rate più alto. L'aspetto più interessante del lavoro di Petrović et al. [18] è rappresentato dal fatto che il loro sistema riesce a processare un nuovo documento proveniente dallo stream, in tempo e spazio costante. Per ottenere tale risultato non solo impiegano l'lsh per ridurre il numero di confronti da effettuare, ma impongono ulteriori limiti per applicare tali tecniche allo streaming dei tweet. Infatti poiché il numero di bucket dell lsh sebbene possa essere alto è un numero finito, e in uno stream

di dati, il numero di documenti che possono ricadere in un bucket potrebbe crescere a dismisura. Per tale ragione pongono un limite sul numero di documenti che può contenere un singolo bucket. Quando un bucket raggiungerà la sua massima capienza, verrà eliminato il documento più vecchio. Questa restrizione sebbene renda il numero di confronti costante, tale costante può divenire piuttosto alta e quindi viene posto un ulteriore vincolo per limitare superiormente il numero di confronti. Bisogna sottolineare però, che se da un lato questi vincoli permettono al sistema di lavorare in setting incrementale, dall'altro andranno ad esacerbare il problema della *fragmentation*, poiché il sistema effettuando per ogni nuovo tweet un numero costante di confronti tenderà a generare un numero maggiore di sotto-eventi. La frammentazione si presenta quando tweets che discutono dello stesso evento sono assegnati a cluster diversi. Questo problema affligge la maggior parte tutti i sistemi che tentano di estrarre documenti dallo streaming fi tweet, tramite il clustering. In particolare impostando una soglia di similarità troppo bassa, si avrà una maggiore probabilità di frammentazione, impostando un valore molto basso invece, può dare origine al problema opposto il *merging* ovvero il problema duale della fragmentation, che si presenta quando tweet di che discutono di eventi diversi sono raggruppati in un unico cluster. Per alleviare il problema della frammentazione, Sankaranarayanan et al [21] suggeriscono, una volta identificati i cluster, di tentare di fondere periodicamente i cluster simili o duplicati. É doveroso sottolineare che tutti i metodi descritti precedentemente, si basano in linea di massima sulla mera similarità testuale fra i tweet, di conseguenza non riusciranno ad identificare pattern di tweet composti da parole sintatticamente diverse ma semanticamente correlate (sinonimia), o al contrario produrranno cluster di tweet composti da termini identici ma, semanticamente non correlati (polisemia).

Capitolo 3

Unsupervised Learning

3.1 DBSCAN

DBSCAN[7](Density-Based Spatial Clustering of Applications with Noise) è un algoritmo di clustering proposto da Martin Ester et. al nel 1996. basato su densità. L'idea di questo algoritmo è che ciascun elemento di un cluster debba avere un vicinato entro un certo raggio ϵ (anche chiamato *Eps*) che contenga almeno un numero minimo di punti (MinPts). Questo algoritmo, in altre parole, stima la *densità* di ciascun punto enumerando gli elementi che ricadono entro un dato raggio ϵ dal punto. Quei punti con una densità superiore ad una certa soglia, $MinPts$, sono classificati come *core objects* .I punti rimanenti invece, saranno etichettati come *noise* se nel loro intorno, entro il raggio stabilito, non contengono nessun core-point, *border* altrimenti.

A partire da questi punti core, vengono definite delle proprietà di connettività verso tutti quei punti che appartengono al eps-vicinato. In seguito verranno date le definizioni formali delle proprietà di connettività fra i punti, che verranno utilizzate per generare i possibili cluster.
definizione 2 (directly density-reachable)**.** Un oggetto p si dice *directly-density-reachable* da un oggetto q rispetto ai parametri *Eps* ed $MinPts$ nell'insieme di oggetti D se:

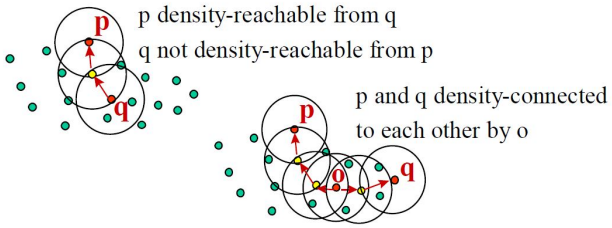


Figura 3.1: Density Reachability e Density Connectivity

1. $p \in Neigh_{eps(Q)}$
2. $|Neigh_{eps(Q)}| \geq MinPts$

definizione 3 (density-reachable)**.** Un oggetto p si dice *"density-reachable"* da un oggetto q rispetto ai parametri Eps ed $MinPts$ nell'insieme di oggetti D, e si indica con $p >_D q$, se esiste una catena di oggetti: $p_1, \ldots, p_n$, $p_1 = p$ $p_n = q$ tali che: $p_i \in D \wedge p_{i+1}$ è *directly density-reachable* da p_i

Questa relazione è l'estensione canonica della relazione di Directly density reachability. Ovvero se abbiamo che un punto p è ddr[1] a q, e q è ddr da r allora diremo che p è *density reachable* da r. Inoltre è una relazione transitiva ma non simmetrica. Sebbene non sia simmetrica, vi sono delle zone nell'isnieme D, in cui questa relazione è simmetrica ovvero per quegli oggetti o nell insieme per cui $|N_{eps(o)}| \geq MinPts$. Due oggetti *border*, ovvero due oggetti che giacciono sul confine di un cluser, probabilmente non saranno density-reachable fra loro in quanto non ci sono abbastanza oggetti nel loro vicinato. Tuttavia, ci sarà necessariamente, un terzo oggetto nel cluster dal quale entrambi questi oggetti border saranno density-reachable.

definizione 4 (density-connected)**.** Un oggetto p si dice *"density-connected"* da un oggetto q wrt Eps ed $MinPts$ nell'insieme di oggetti D se esiste un oggetto $o \in D$ tale che sia p che q siano *density reachable* da o. La relazione di density-connectivity è simmetrica, la figura 3.1 mostra la le definizioni su descritte in uno spazio bi-dimensionale.

Proprio grazie alla proprietà di density connectivity, si potrà dare la definizione di cluster ovvero un insieme massimale di oggetti "densamente connessi".

[1] ddr=directly density reachable

definizione 5 (cluster)**.** Sia D un insieme di oggetti. Si defnisce *"cluster"* C, wrt Eps ed $MinPts$ nell'insieme di oggetti D, un sottoniseme non vuoto di D, che soddisfa le seguenti condizioni :

1. *Massimalità*: $\forall p, q \in D$, se $p \in C \wedge q >_D p$ wrt MinPts e Eps, allora anche $q \in C$

2. *Connettività*: $\forall p, q \in C$, p è *"density-connected"* a q wrt MinPts e Eps

definizione 6 (noise)**.** Siano $C_1, \ldots, C_k$ tutti i cluster wrt Eps e , ovvero come un insieme di oggetti densamente connessi che è massimale rispetto MinPts in D. Si definisce *noise* l'insieme di oggetti D che non appartengono a nessun cluster C_i, noise=$\{p \in D | \forall i : p \notin C_i\}$.

Questo algoritmo presenta due proprietà fondamentali che ne permettono una computazione efficiente:

1. Dato un qualsiasi core-object, l'insieme dei punti density reachable da tale punto (w.r.t Eps,MinPts) costituirà un cluster.

2. Si consideri un cluster C, ciacun elemento di C è density-reachable da un ogni core-object in C. Ciascun cluster sarà quindi, univocamente identificato da uno qualsiasi dei suoi core-object.

L'algoritmo di DBSCAN, al fine di creare un cluster,parte da un punto arbitrario p e ritrova tutti i punti da esso density-reachable rispetto ai parametri Eps e $MinPts$, effettuando region-query [2] prima per p ed eventualmente per i vicini di p diretti ed indiretti. Se l'oggetto p è un core-object tale procura ci darà un cluster, altrimenti l'oggetto p sarà considerato come NOISE, e si passerà al oggetto successivo. Se p è un border-point sarà successivamente ri-etichettato quando, verrà considerato un core object dal quale p è density reachable. Le implementazioni classiche di DBSCAN fanno uso di indici spaziali come R-tree o X-tree, che consentono di ritrovare il viciniato di ciacun punto (region-query) in maniera efficiente $O(logn)$. Nel peggiore dei casi, DBSCAN visita ogni punto del dataset, ovvero esegue una region query per ciascun punto, ciò determina una complessità $0(n^2)$ dove n rappresenta

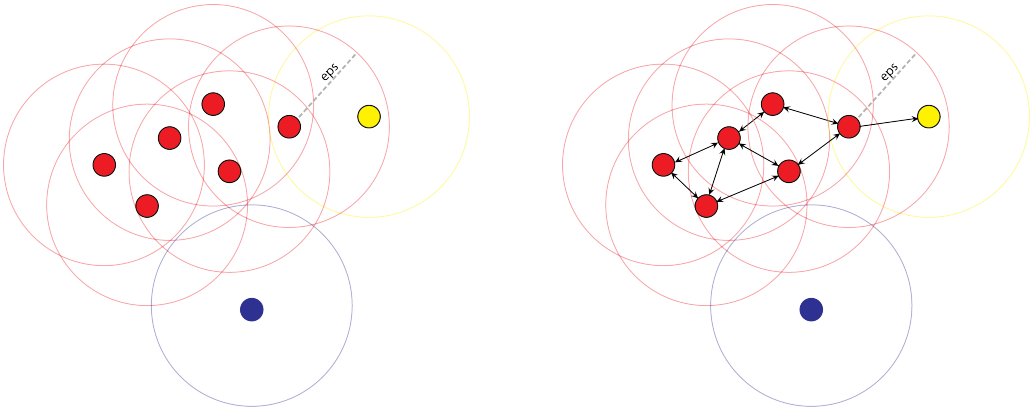
[2] una region-query di un punto p è la ricerca dei punti che ricadono nel viciniato di p

il numero di punti nel dataset. Se si utilizza una struttura indicizzata come un R-tree, è possibile passare ad una coplessità $0(nlogn)$. Tali strutture indicizzate però, non riescono a scalare all'aumentare della dimensionalità dei dati: la performance delle region query passa da O(logn)to O(n), facendo degenerare, cioè la complessità temporale di dbscan in $O(n^2)$ rendendo poco adatto questo algoritmo al problema del clustering dei tweets. Nel lavoro di Guo et al. [23] vien proposto una variante dell'algoritmo DBSCAN che fa uso dell'LSH, al fine di ritrovare i nearest-neighbors di un punto, riducendo sensibilmente la complessità dell'algoritmo.

L'algoritmo DBSCAN riesce a generare dei cluster anche con un elevata presenza di rumore. Inoltre grazie alla definizione di "density-connected"4, riesce ad identificare cluster di forme arbitrarie. Inoltre, a differenza di altri algoritmi come k-means, non necessita di conoscere a priori il numero di cluster. Tutte queste caratteristiche rendono questo algoritmo particolarmente interessante nella scoperta di cluster in questo lavoro di tesi poiché: i messaggi di un microblog come Twitter contengono un'alta percentuale di rumore, e nel task di event-detection non è possibile sapere a priori il numero di eventi.

3.1.1 DBSCAN come ricerca di componenti connesse

Il problema del clustering, può essere ricondotto al problema della ricerca di componenti connesse in un grafo. In un grafo, una *componente connessa* è un sotto-grafo in cui ogni coppia di vertici è connessa da un cammino, e il sottografo non è connesso a nessun altro vertice del grafo di partenza. Per ottenere una corrispondenza fra una componente ed un cluster secondo dbscan, sarà sufficiente far sì che per ogni coppia di punti density-connected, esista un cammino che li colleghi. In figura 3.2b è mostrato come è possibile rappresentare un grafo non diretto a partire da connessioni di densità: per ogni coppia di oggetti di directly density reachable a, b con a core-object, verrà creato un arco (a, b). Talvolta però, un border-object può essere nel vicinato di più di un core-object anche appartenenti a cluster distinti, come



(a) MinPts=3 **(b)** Esempio di grafo di densità

Figura 3.2: rossi=core-objects, gialli=border, blue=noise

mostrato in figura 3.3. Se fossero lasciati entrambi gli archi verso il border-object, verrebbe identificata un'unica componente connessa. Lasciare questi archi renderebbe l'algoritmo molto più sensibile al rumore, poiché un solo punto (magari un noise) potrebbe determinare la fusione di più cluster.

Per evitare tali problematiche, bisogna costruire un grafo i cui nodi sono sono soltanto i *core-objects*, e verrà aggiunto un arco fra due nodi solo se la loro distanza è minore di *eps*. In maniera più formale il grafo è cosi definito:

$G = (V, A)$ dove :

$V = \{x \in D \mid Card(Neigh_{eps(x)}) \geq MinPts\}$

$A = \{(x, y) \in V \mid d(x, y) \leq eps\}$

A partire da questo grafo G verranno identificate tutte le componenti connesse che corrispondo ai cluster, individiuati però solo a partire dai nodi cores. Bisognerà quindi eseguire un ulteriore step per assegnare gli oggetti border ad un cluster con cui sono connessi. L'algoritmo originale nel caso in cui un oggetto border ricada nel vicinato di più core objects, ciascuno dei quali di un cluster differente, lo assegnerebbe in maniera casuale ad uno di questi cluster. In questo caso è possibile adottare tre strategie diverse:

- scegliere in maniera casuale un cluster

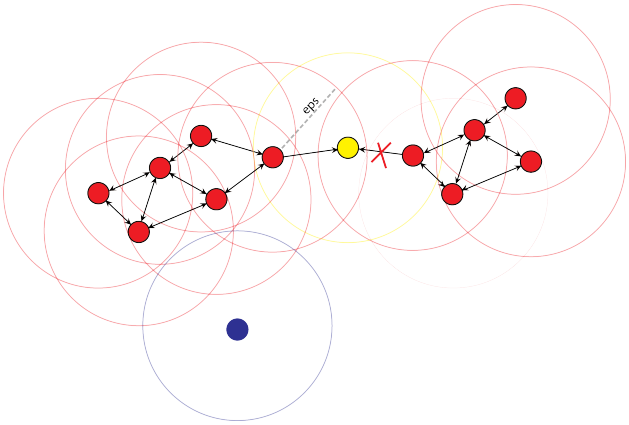


Figura 3.3: border object density reachable da due core-object

- scegliere il cluster con cui il border object ha maggiore connettività
- scegliere il cluster più vicino al border object

3.2 Local Sensitive Hashing

Un problema fondamentale che molti task di Data Mining devono affrontare è quello di esaminare i dati per trovare item "simili". Molto spesso questo problema è risolto cercando il Nearest Neighbor (o i primi k-nearest neighbor) di un oggetto in qualche spazio metrico R^d . Nel caso di poche dimensioni (d=2, d=3) questo problema, si riesce a risolvere in maniera piuttosto efficiente (O(log n)) mediante l'utlizzo di strutture dati come K-D-Tree o R-tree che partizionano lo spazio delle feature. Tuttavia al crescere delle dimensioni (curse of dimensionality), queste strutture basate sul partizionamento dello spazio, degradano in una ricerca lineare [22] passando quindi ad una complessità quadratica. Per rendere meglio l'idea supponiamo di voler calcolare i documenti più simili in una collezione composta da un milione di documenti, si avrebbero quindi $\binom{100000}{2}$ coppie di cui calcolare la similarità. Se ci si impiega un microsecondo per ogni calcolo di similarità, sarebbero necessari quasi sei giorni per valutare tutte queste coppie. Se l'obiettivo è proprio il calcolo delle similarità di ogni possibile coppia, l'unico modo per ridurre il tempo necessario è usare una qualche forma di parallelismo. Tut-

tavia spesso, come nel caso del NNS-problem, si è interessati solo alle coppie più simili o più in generale a quelle coppie la cui similarità è al di sopra di una determinata soglia. É quindi sufficiente considerare solo quelle coppie che "probabilmente" sono simili, piuttosto che considerarle tutte. Proprio a tale scopo, se la dimensionalità dei dati è piuttosto alta, è possibile usare la tecnica del *Local Sensitive Hashing* (LSH) [12] [9]. L'idea chiave dell'LSH è quella di eseguire l'hasing dei punti attraverso molte funzioni di hashing, in modo tale che per ogni funzione la probabilità di collisione sia più alta per i punti vicini fra loro, rispetto ai punti più distanti. Dopo aver eseguito l'hashing tutte le coppie che sono state mappate nello stesso "bucket" possono essere considerate come *coppie candidate*, solo queste coppie saranno valutate per determinare quali sono realmente simili. Quello che ci si augura è che, le coppie di item fra loro dissimili non vegano mai mappate verso lo stesso bucket,tali coppie rappresentano infatti dei *false positive*. Allo stesso tempo ci si auspica che tutte (o la maggior parte) delle coppie realmente simili siano mappate nel medesimo bucket, le coppie che simili che non sono ricadute nello stesso bucket rappresentano i *false-negative*

3.2.1 Teoria dell'LSH

L'LSH è un framework per costruire strutture dati che permettono di ricercare i "near neighbor" all'interno di una collezione di vettori altamente dimensionali. Dato un insieme P contenente vettori D-dimensionali, l'obiettivo è di costruire una struttura che consenta di ritrovare, per un qualsiasi query point q, tutti quei punti che giacciono in un raggio di distanza cR (dove c rappresenta un fattore di approssimazione > 1), o più formalmente gli R-*near neighbors* di q. Data una funzione di hashing h diremo che una coppia di oggetti (x, y) sarà una *candidate* se $h(x) = h(y)$. Sia $d(\cdot, \cdot)$ una qualche funzione definita su un insieme di oggetti S

definizione 7 (Locality-sensitive hashing)**.** Una famiglia di funzioni H si dice $(d_1, d_2, p_1, p_2) - Sensitive$ se $\forall (x, y) \in S,\ \forall h \in H$

- *se* $\quad d(x, y) \leq d_1 \Rightarrow P_H[h(x) == h(y)] \geq p_1,$

- *se* $d(x,y) \geq d_2 \Rightarrow P_H[h(x) == h(y)] \leq p_2$

dove $p_1 > p_2$

A differenza di algoritmi di hashing classici dove si cercano di evitare delle collsioni per item diversi, in questo caso si cerca di massimizzare la probabilità di collisione per item vicini. Dato un oggetto q come query, verranno restituiti gli oggetti che giacciono nello stesso bucket $h(q)$

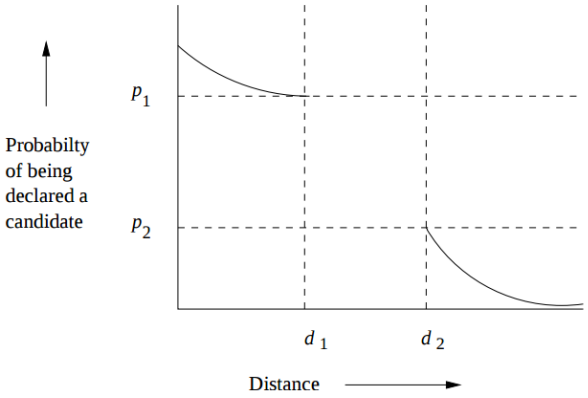
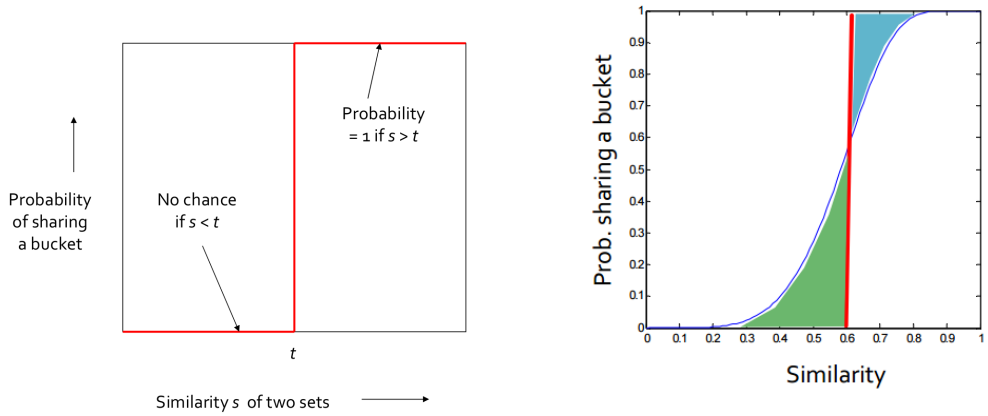


Figura 3.4: esempio di una funzione $(d_1, d_2, p_1, p_2) - sensisitve$

La figura 3.4 mostra un la probabilità che una possibile funzione di una famiglia $(d_1, d_2, p_1, p_2) - sensisitve$ dichiari una coppia come candidata.

Data una famiglia di funzioni hash $(d_1, d_2, p_1, p_2) - sensisitve$ H si può costruire una nuova famiglia di funzioni *H'* applicando la *AND-construction* su H come segue: ciascuna funzione di H' sarà data dalla concatenazione da r funzioni di H : $\{h_1....h_r\}$. Allora per ogni funzione h in *H'* diremo che: $h(x) = h(y)$ se $h_i(x) = h_i(y)$ per ogni $i...r$. Dato che le funzioni h_i sono scelte in maniera indipendente da H, possiamo asserire che *H'* è una famiglia $(d_1, d_2, p_1^r, p_2^r) - sensisitve$. Se invece costruiamo una nuova famiglia di funzione, per mezzo di una disgiunzione di b elementi di H (*OR-construction*), passeremo da una famiglia $(d_1, d_2, p_1, p_2) - sensisitve$ ad una $(d_1, d_2, 1-(1-p_1)^b, 1-(1-p_2)^b) - sensisitve$ *H'*. In questo caso diremo che: $h(x) = h(y)$ se esiste un valore di i tale che $h_i(x) = h_i(y)$ Se p è la probabilità che elemento di H dichiari una coppia (x, y) come candidata,



(a) s-curve ideale **(b)** s-curve r=5,b=10

Figura 3.5: esempi di S-Curve

allora $1-p$ è la probabilità che non lo sia. Avendo b funzioni $h_1, ..., h_b$ avremo che la probabilità che nessuna di esse dichiari la coppia come candidata, pari a $(1-p)^b$, e di conseguenza $1-(1-p)^b$ sarà la probabilità che almeno una fra le b funzioni dichiarerà la coppia come candidata. É facile notare che la *And-Construction* fa decrescere tutte le probabilità, mentre la *OR-construction* ha l'effetto opposto.É però possibile, applicare in cascata queste due tecniche AND,OR in modo tale che la probabilità p_2 sia più bassa possibile, mentre p_1 sia quanto più possibile vicino ad 1, ottenendo a una famglia $(d_1, d_2, 1-(1-p_1^r)^b, 1-(1-p_2^r)^b) - sensisitve$. Scegliendo in maniera accurata i valori di r e di b sarà possibile controllare l'andamento della funzione di probabilità $1-(1-p^r)^b$. Tale funzione, una volta fissati i parametri r e b, descrive una *S-curve*. Come per qualsiasi *S-curve*, è possibile determinare il punto fisso, ovvero quel valore di t che rimane inalterato dopo aver applicato, la funzione $p = 1-(1-p^r)^b$. Tale valore può essere approssimato come $t \approx (1/b)^{\frac{1}{r}}$. Al di sotto di tale soglia, il valore di probabilità decresce,al di sopra cresce.Quindi, se se scegliamo la probabilità più alta p_1 al di sopra della soglia t, e p_2 al di sotto, avremo che il valore p_2 è ridotto e al contempo il valore p_1 viene innalzato. Bisogna quindi impostare i valori di b, r in modo tale che il punto fisso della funzione, corrisponda proprio al valore di similarità tale per cui due elementi li possiamo considerare "simili". La figura 3.5a mostra il caso ideale: dopo un valore di soglia la propabilità che due una

coppia diventi candidata diviene immediatamente pari a 1, al di sotto pari a zero. Nella figura 3.5b viene mostrato invece il caso in cui $r = 5, b = 10$, l'area verde indica il false-positive rate, mentre l'area blue indica il false negative rate. Bisogna quindi, effettuare un tuning dei parametri b, r per cercare di ritrovare la maggior parte delle coppie di elementi realmente simili, e al contempo avendo pochi false-positive. Bisogna tenere a mente, però , che i false-positive possono essere filtrati con una fase di post-processing valutando la similarità reale fra le coppie restituite, i false negative, invece non possono essere in alcun modo recuperati, poiché rappresentano quelle coppie di item la cui distanza è bassa, ma che non sono state mappate nello stesso bucket.

3.2.2 LSH-Cosine

In questo lavoro di tesi poiché si stanno analizzando documenti testuali, verrà utilizzata una famiglia di funzioni di hashing local-sensitive per la distanza del coseno. In particolare sarà adottato lo schema di lsh proposto da Charikar[6] in cui la la probabilità che due punti collidano (ovvero che siano mappati verso lo stesso bucket), è proporzionale al coseno dell'angolo fra di loro. Data una collezione di vettori in R^d viene definita una famiglia di funzioni hashing come segue: si sceglie un vettore random $\vec{r}$ le cui componenti sono presse da una distribuzione Gaussiana. Proprio grazie a questo vettore random verrà definita una funzione di hashing come segue:

$$h_{\vec{r}}(\vec{u}) := \begin{cases} 1 & \text{se } \vec{r} \cdot \vec{u} \geq 0, \\ 0 & \text{se } \vec{r} \cdot \vec{u} \leq 0, \end{cases} \tag{3.1}$$

La figura 3.7 mostra l'interpretazione geometrica dell'equazione 3.1, ovvero valutare il segno del prodotto interno fra il vettore che definisce la funzione di hashing, e un un vettore item, equivale a stabilire se l'item è al di sotto o al di sopra dell'iperpiano identificato dalla funzione. Costruendo in questa

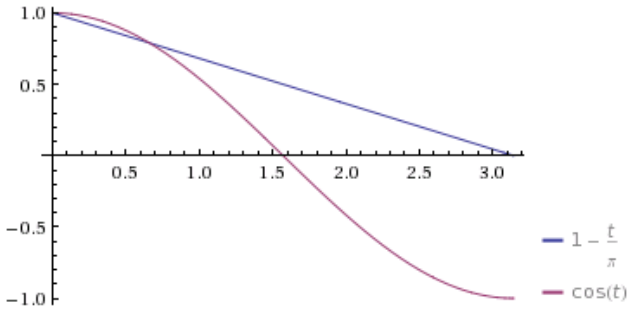


Figura 3.6: approsimazione sim. coseno

maniera una funzione di hashing si avrà che, per due vettori $\vec{u}, \vec{v}$,

$$P[h_{\vec{r}}(\vec{u}) = h_{\vec{r}}(\vec{v}))] = 1 - \frac{\theta(\vec{u}, \vec{v})}{\pi} \tag{3.2}$$

Secondo il teorema3.2 [10], la probabilità che un iperpiano random, separi due vettori è direttamente proporzionale all'angolo fra di essi. In figura 3.6 si può notare che per piccoli angoli (non vicini all'angolo retto), $1 - \frac{\theta}{\pi}$ è una buona approssimazione per $\cos(\theta)$. Inoltre, dall'equazione 3.2 si ha che

$$cos(\theta(u, v)) = cos((1 - P[h_r(u) = h_r(v)])\pi) \tag{3.3}$$

Grazie a questa equazione, è possibile stimare la probabilità del coseno attraverso il risultato dell'applicazione delle funzioni di hashing. Guardando con più attenzione si può notare che

$$P[h_r(u) = h_r(v)]) = g1 - hammingDistance(h_r(u), h_r(v))/r$$

In pratica, questa uguaglianza permette di passare dal problema del calcolo della similarità del coseno fra due vettori altamente dimensionali, al problema del calcolo della distanza di hamming fra due stringhe.

Anche questa famiglia di funzioni di hashing può essere aplificata con la tecnica AND-OR precedentemente descritta.

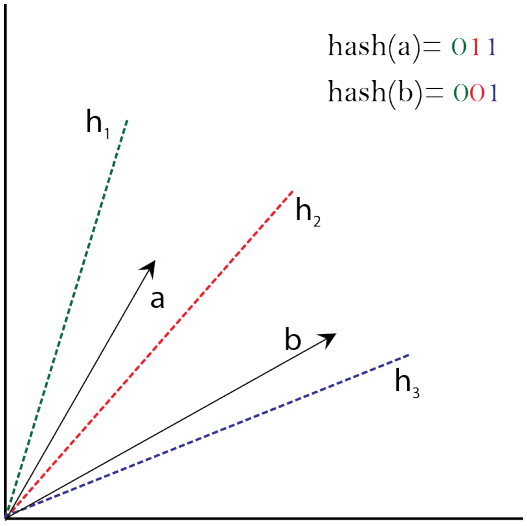


Figura 3.7: funzione di hashing

Capitolo 4

Sperimentazione

In questo capitolo si descriverà la progettazione e l'esecuzione della sperimentazione. Si partirà pertanto dalla descrizione dati su cui quest'ultima è stata effettuata, proseguendo con la scelta delle modalità di esecuzione più interessanti e concludendo con una serie di tabelle e grafici contenenti i risultati ottenuti, opportunamente commentati. L'obiettivo di questa prima sperimentazione è valutare l'efficacia del sistema analizzato al fine di comprendere le cause di un eventuale successo o insuccesso delle tecniche utilizzate e quindi decidere se proseguire gli studi in questa direzione o definire strategie alternative. Il corpus di documenti utilizzato[16]. per testare le performance della soluzione proposta, è stato costruito proprio per la valutazione del task di event detection. Questo corpus ricopre un periodo di quattro settimane ed è composto circa 120 milioni di tweets. A partire da questi tweets, gli autori del corpus hanno generato un pool di eventi "candidati "attraverso alcuni approcci dello stato dell'arte e Wikipedia. Successivamente,attraverso crowdsourcing, sono stati raccolti i giudizi di rilevanza rispetto di questi eventi candidati, solo dopo aver fornito una propria definizione di evento. Al termine di tale processo sono stati raccolti i giudizi di 150.000 tweet, suddivisi in 500 eventi.

In seguito è fornita una descrizione maggiormente dettagliata di tale dataset.

4.1 Dataset per la valutazione

Sebbene il task di event detection sia un'area molto fervida di ricerca, vi sono, ad oggi, pochi corpus disponibili per valutare le performance di un sistema su larga scala. Creare un tale corpus, a causa della srande mole di dati, richiede sia molto tempo che risorse. Inoltre, a causa dei termini di servizio di Twitter [1], non è possibile includere in questi corpus il contenuto dei tweet, ma solo gli identificativi di questi ultimi.

Al fine di testare il sistema prodotto, è stato adoperato un corpora costruito proprio per la valutazione di sistemi di scoperta di eventi a partire da Twitter [16]. In questo corpus sono stati raccolti tweets a partire dal 10/10/2012 al 7/11/2012, tramite le API streaming di Twitter. La scelta di questo intervallo temporale non fu casuale, ma è motivata dal fatto che, in quel periodo, erano previsti eventi molto importanti come le elezioni presidenziali degli Stati Uniti e l'uragano Sandy. Gli autori del dataset, per poter raccogliere dei giudizi di rilevanza in merito ad eventi, hanno innanzitutto fornito il loro concetto di evento come segue:
definizione 8. Un evento è qualcosa di **significante** che avviene in un luogo e tempo specifico.

Rispetto alla definizione1 data dal TDT, viene aggiunto il vincolo di *significatività* di un evento, per scartare dal pool di eventi candidati, quelli che si riferiscono ad eventi banali o contenenti status prettamente personali. Nel task del TDT infatti, è ragionevole assumere che tutti i documenti siano rilevanti, in quanto estratti da newswire.
definizione 9. Qualcosa è *significante* se può essere discusso dai media.

Gli autori sottolineano, però, che non è necessario che qualcosa sia discusso dai media per essere considerato un evento, ma utilizzano tale definizione solo come livello di significatività necessario affinché si possa parlare di un evento.

[1] https//dev.twitter.com/overview/terms/agreement-and-policy: *"If you provide an API that returns Twitter data, you may only return IDs (including tweet IDs and user IDs)."*

Sono stati applicati i seguenti filtri

- **language-filter** : sono stati filtrati solo i tweet di lingua inglese [2] , utilizzando una libreria java che consente una la scoperta automatica della lingua.
- **spam-filter** : Al fine di rimuovere parte dello spam presente su twitter, sono state applicate le seguenti regole empiriche [4]
  - tweet con più di tre hashtag
  - tweet con più di due url
  - tweet con più di tre user-mentions

Dopo l'applicazione di questi filtri sono stati raccolti 120 milioni di tweets. Di questi, circa il 30% era costituito da *retweet* (quasi 40 milioni). Poiché i retweet non sono altro una copia di un tweet di un altro utente, i creatori del corpus non hanno incluso questi ultimi nei giudizi di rilevanza.

4.1.1 Generazione Eventi Candidati

Piuttosto che creare manualmente una lista di eventi, gli autori hanno utilizzato due approcci della letteratura (*detection-approach*) e Wikipedia Events Curent Portal (*curated-approach*) [3].

I sistemi di scoperta di eventi della letteratura usati sono i seguenti:

- l'approccio basato su LSH proposto da Petrović [18]
- l'approccio basato su Cluster-Summarization proposto da Aggarwal e Subbian [1]

I creatori del dataset hanno scelto questi due approcci poiché sono document-pivot, ovvero producono cluster di document (piuttosto che cluster di termini: è quindi più semplice generare dei giudizi di rilevanza),ed inoltre entrambi

[2] https://code.google.com/archive/p/language-detection/
[3] https://en.wikipedia.org/wiki/Portal:Current_events

gli approcci scelti, sono abbastanza efficienti e riescono a processare grandi moli di dati. L'algoritmo proposto da Petrović è stato lanciato su tutto il corpus , utilizzando dei parametri molto simili a quelli proposti nell'articolo di riferimento [18] ovvero:

1. 13 bits per chiave (numero di hash functions)
2. 70 hash tables
3. 0.45 come distanza massima

A differenza dell'articolo orginale, sono stati considerati i clusters che crescono più velocemente su base oraria (piuttosto che ogni 100K). Per ogni ora, è stata prodotta una lista di cluster. Tale lista è stata ordinata in base al numero di utenti distinti, di ciascun cluster, e i cluster aventi bassa entropia sono posizionati in coda alla lista. Considerare una lista così prodotta avrebbe generato un numero troppo elevato di eventi candidati. Per tale ragione gli autori hanno rimosso tutti quei cluster il cui valore di entropia ricade al di fuori dell'intervallo ottimale [3.5,4.25][18], producendo così una lista di 1340 eventi candidati. L'algoritmo di Aggrawal e Subbian, invece, richiede a priori il numero di cluster in cui suddividere il corpus. I creatori del dataset hanno scelto k=1200 in quanto questo parametro permetteva loro di clusterizzare l'intera collezione in un tempo ragionevole (4 giorni). Il parametro λ è stato impostato pari a 0, dando cioè, il peso massimo alla similarità testuale). Sono stati rimossi, empiricamente, i cluster aventi meno di 30 tweet e con un basso grow-rate ($\alpha < 12$), producendo in tal modo una lista di 1097 eventi candidati.

Attraverso il portale degli eventi correnti, Wikipedia, mantiene una lista di eventi giornalieri, disseminati su tutto il globo. Ad ogni evento è associato un link ad un articolo contenente degli approfondimenti in merito a tale evento (ciascun evento di tale lista quindi, combacia perfettamente con la definizione 8 data precedentemente). Di ogni evento viene fornita una breve descrizione, la data dell'evento, e uno o più link ad articoli rilevanti.

Questo approccio è completamente diverso rispetto a quelli precedentemente

illustrati. In questo caso, infatti, si esegue il processo opposto: si parte da una lista di eventi e l'obiettivo è quello di associare a ciascuno di essi un insieme di tweet rilevanti. Per far ciò il primo step è stato quello di indicizzare l'intero corpus ramite Lucene 4.2, dopo aver applicato delle operazioni di nlp come rimozione di stop-word, URLs e i prefissi di hashtag e mentions (#,@). Per ogni evento (dei 468 eventi del portale Wikipedia), è stato interrogato il corpus utilizzando come query la descrizione dell'evento. Per ciascuna query sono stati ritrovati i top 2000 documenti da una finestra di 72 ore, centrata sulla data dell'evento.

Generazione Giudizi Rilevanza

La valutazione degli eventi candidati è stata effettuata in crowd-sourcing utilizzando Amazon Mechanical Turk. L'obiettivo di questa valutazione era decidere quale degli eventi candidati, combaciava con la definizione8 di evento data in precedenza. Per valutare gli eventi candidati prodotti dagli approcci di event-detection ad ogni annotatore è stato sottoposto un questionario. In particolare ad ogni annotatore era richiesto di leggere 13 tweet scelti casualmente da ogni cluster, e gli si chiedeva innanzitutto se questi discutessero dello stesso topic. In caso affermativo, veniva posta un ulteriore domanda ovvero se questi discutessero o meno di un "evento". Se rispondevano positivamente, questi dovevano leggere i tweeet del cluster ed etichettarli come rilevanti o meno.

Nel caso, invece, dei risultati prodotti dall'approccio curated, si sà già a priori che ciascuno dei candidati è un evento. Agli annotatori era solo richiesto di etichettare ciascun tweet ritrovato come rilevante o meno.

I risultati prodotti dai metodi di event-detection sono stati considerati come eventi se almeno il 50% degli annotatori gli ha etichettati come eventi. Per il metodo basato su LSH sono stati filtrati 382 eventi, 52 per l'apporoccio CS. I candidati prodotti dall'apporccio curated, sono stati considerati come eventi se hanno prodotto almeno un tweet rilevante, producendo cosi 361 eventi.

Ciascuno dei tre metodi utilizzato ha prodotto insiemi di eventi diversi, ma non disgiunti, esiste cioè un overlap molto i vari risultati. Ad esempio il metodo basato su LSH ha prodotto 40 cluster di eventi per l'evento relativo al terzo dibattito presidenziale. Per questo è stato applicato un algoritmo di clustering sui risultati dei tre metodi al fine di fondere quelli fra loro più simili. A termine di questa operazione, sono stati prodotti 506 cluster che corrispondono quindi ad eventi.

4.1.2 Raccolta dati

Di questi 120 milioni di tweets, sono stati resi pubblici solo gli (id dei tweet e degli utenti di tali tweet). É necessario quindi, un ulteriore step per ritrovare il contenuto di tali tweet. Avendo gli identificativi dei tweets, esistono due metodologie per ritovare il contenuto: le API di twitter, il crawling di twitter.com. Utilizzare le API di twitter fornendo come query gli id [4], permette di ottenere non solo il contenuto testuale, ma anche ulteriori meta-dati come eventuali hashtag, urls, mentions, in formato JSON. Tuttavia vi è un limite orario di 150 richieste [5], quindi scaricare un corpus di grandi dimensioni, come nel nostro caso, richiederebbe troppo tempo. Per tale ragione, si è adottato un crawler fornito pubblicamente dal TREC Microblog Task [6], che consente di aggirare questo limite. A termine di questa fase di crawling sono stati raccolti circa 15 milioni di tweets, di cui circa 10 milioni sono retweets. Poiché i retweet non sono sati inclusi nei giudizi di rilevanza, si è deciso di scartarli in quanto non influenti ai fini della valutazione. Bisogna sottolineare che, molti tweet non sono stati ritrovati poiché, utenti che avevano pubblicato status raccolti nel corpus, hanno eliminato i loro tweets o perfino il loro profilo.

[4] https://dev.twitter.com/rest/reference/get/statuses/
[5] in ogni richiesta si possono richiedere 100 id
[6] https://github.com/myleott/twitter-corpus-tools

Bibliografia

[1] Charu C. Aggarwal and Karthik Subbian. *Event Detection in Social Streams*, chapter 53, pages 624–635.

[2] James Allan, editor. *Topic Detection and Tracking: Event-based Information Organization*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[3] Hila Becker, Mor Naaman, and Luis Gravano. Beyond trending topics: Real-world event identification on twitter. In *Proceedings of the Fifth International Conference on Weblogs and Social Media, Barcelona, Catalonia, Spain, July 17-21, 2011*, 2011.

[4] Fabrício Benevenuto, Gabriel Magno, Tiago Rodrigues, and Virgílio Almeida. Detecting spammers on twitter. In *In Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS*, 2010.

[5] Carlos Castillo, Marcelo Mendoza, and Barbara Poblete. Information credibility on twitter. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 675–684, New York, NY, USA, 2011. ACM.

[6] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 380–388, New York, NY, USA, 2002. ACM.

[7] Martin Ester. A density-based algorithm for discovering clusters in large spatial databases with noise. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 226–231. AAAI Press, 1996.

[8] Michael Gamon, Sumit Basu, Dmitriy Belenko, Danyel Fisher, Matthew Hurst, and Arnd Christian König. Blews: Using blogs to provide context for news articles. In *2nd AAAI Conference on Weblogs and Social Media (ICWSM 2008)*. American Association for Artificial Intelligence, April 2008.

[9] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[10] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, November 1995.

[11] Jonathan Hurlock and Max L. Wilson. Searching twitter: Separating the tweet from the chaff. In *Proceedings of the Fifth International Conference on Weblogs and Social Media, Barcelona, Catalonia, Spain, July 17-21, 2011*, 2011.

[12] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[13] Jon Kleinberg. Bursty and hierarchical structure in streams. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 91–101, New York, NY, USA, 2002. ACM.

[14] Kyumin Lee, Brian David Eoff, and James Caverlee. Seven months with the devils: A long-term study of content polluters on twitter. In

Proceedings of the Fifth International Conference on Weblogs and Social Media, Barcelona, Catalonia, Spain, July 17-21, 2011, 2011.

[15] Gang Luo, Chunqiang Tang, and Philip S. Yu. Resource-adaptive real-time new event detection. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 497–508, New York, NY, USA, 2007. ACM.

[16] Andrew J. McMinn, Yashar Moshfeghi, and Joemon M. Jose. Building a large-scale corpus for evaluating event detection on twitter. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 409–418, New York, NY, USA, 2013. ACM.

[17] R. Papka. On-line new event detection, clustering, and tracking title2:. Technical report, Amherst, MA, USA, 1999.

[18] Saša Petrović, Miles Osborne, and Victor Lavrenko. Streaming first story detection with application to twitter. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 181–189, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[19] Swit Phuvipadawat and Tsuyoshi Murata. Breaking news detection and tracking in twitter. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 03*, WI-IAT '10, pages 120–123, Washington, DC, USA, 2010. IEEE Computer Society.

[20] Gerard Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[21] Jagan Sankaranarayanan, Hanan Samet, Benjamin E. Teitler, Michael D. Lieberman, and Jon Sperling. Twitterstand: News in tweets. In *Proceedings of the 17th ACM SIGSPATIAL International Conference*

on Advances in Geographic Information Systems, GIS '09, pages 42–51, New York, NY, USA, 2009. ACM.

[22] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[23] Y. P. Wu, J. J. Guo, and X. J. Zhang. A linear dbscan algorithm based on lsh. In *Machine Learning and Cybernetics, 2007 International Conference on*, volume 5, pages 2608–2614, Aug 2007.

[24] Yiming Yang, Tom Pierce, and Jaime Carbonell. A study of retrospective and on-line event detection. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 28–36, New York, NY, USA, 1998. ACM.