Ahi mè ...

Indice

Elenco delle tabelle

Elenco delle figure

Introduzione

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 on 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. 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[2] o emoticons.

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

[2] Out Of Vocabolary

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 [**?**]. 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. 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*

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 in-

iziarono a far diffondere la notizia [3].Anche nel caso del terremoto in Abruzzo del 6 aprile 2009, gli utenti Twitter hanno segnalato la notizia prima dei media tradizionali.

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

Capitolo 1

Stato dell'Arte

1.1 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* [**?**], 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 [**?**], mentre

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

nella seconda si studia la distribuzione delle singole parole e scoprono nuovi eventi raggruppando le parole [**?**] Come evidenziato da [**?**] 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* [**?**] 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 [**?**] 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 [**?**, **?**] 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 [**?**] è quella di modificare il vocabolario dei termini in maniera incrementale e modificare l'IDF ogni qual volta viene aggiunto un nuovo documento.

$$idf_t(w) = log_2 \left(\frac{N_t}{df_t(w)} \right) \tag{1.1}$$

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* [**?**, **?**] 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 [**?**]. 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 [**?**]. L'assunzione fatta da queste tecniche è che alcune parole avranno un incremento di utilizzo repentino quando accade un evento. Nel lavoro di [**?**] 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 [**?**] 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.

1.2 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 che hanno è il 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) [**?**] e rumors [**?**]. Inoltre le caratteristiche di Twitter e la sua poplarità sono molto allettanti per spammers e altri e altri content polluters [**?**] 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 appropri-

ate per i tweet. Un altro lavoro interessante che si colloca in quest'area è TwitterStand [**?**]: 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 [**?**]. 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 [**?**] 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. AL fine di permettere di eseguire il task di scoperta di eventi su larga scala invece, in [**?**] è stato ideato un algoritmo che fa ricorso al *Local Sensitive Hashing*[**?**] 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.

$$a \; links \; b \iff 1 - cos(a, b) < t \tag{1.2}$$

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. [**?**] 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 [**?**] 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 2

Unsupervised Learning

2.1 DBSCAN

L'idea "chiave"dell'algoritmo DBSCAN[**?**] è che ciascun elemento di un cluster debba avere un vicinato entro un certo raggio ϵ (Eps) che contenga almeno un numero minimo di punti (MinPts). Questo algoritmo, in altre parole, stima la *densità* di ciascun punto, per mezzo della cardinalità del suo "vicinato"entro un raggio predefinito *Eps*. Un punto sarà definito come *core-object* se la densità del suo vicinato supera una data soglia $MinPts$. Un punto q si dice "directly density-reachable"da un core-object p, se q ricade nell'eps-vicinato di p. La chiusura transitiva di questa proprietà, è detta density-reachability. Due punti p e q si diranno invece, *density-connected*, se esiste un terzo oggetto o dal quale entrambi sono density-reachable. Grazie a quest'ultima proprietà è possibile definire un cluster, ovvero come un insieme di oggetti densamente connessi che è massimale rispetto alla proprietà di density-reachability. Il noise, invece, sarà dato dall'insieme di punti che non appartengono a nessuno di questi cluster. In seguito verranno date le definizioni in maniera più formale:

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:

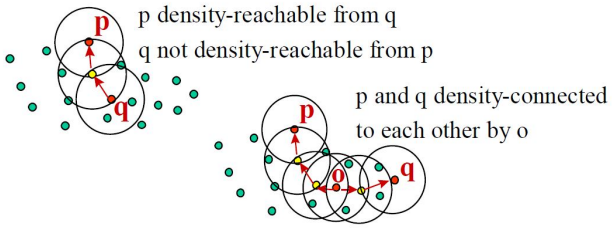


Figura 2.1: Density Reachability e Density Connectivity

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: $p_1, \ldots, p_n$, $p_1 = pp_n = q$ 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 Directly-density-reachability, inoltre è una relaione 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 **??** 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".

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 [1] 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 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

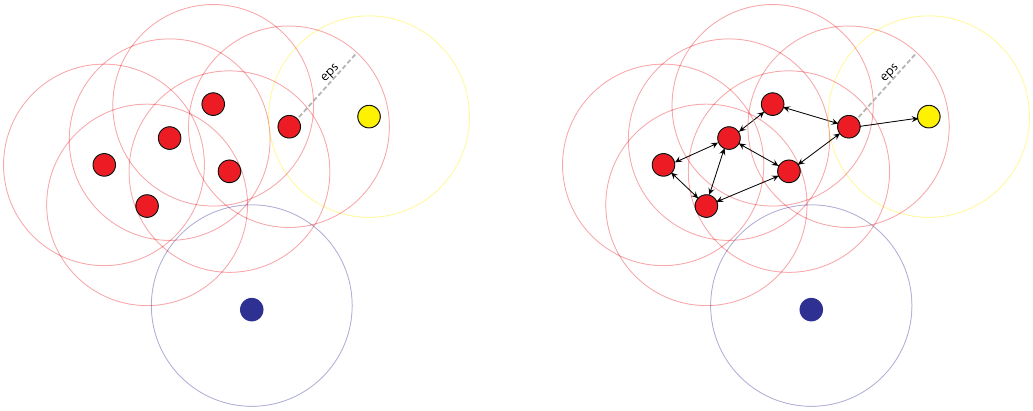
[1] una region-query di un punto p è la ricerca dei punti che ricadono nel viciniato di p

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. [**?**] 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"**??**, 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.

2.1.1 DBSCAN come ricerca di componeti 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 **??** è mostrato come è possibile rappresentare un grafo-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), se anche b è un core object sarà creato un ulteriore arco da b ad a : (b, a). Talvolta però, un border-object può essere nel vicinato di più di un core-object anche appartenenti a cluster distinti, come mostrato in figura **??**. Se fossero lasciati entrambi gli archi verso il border-object,



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

Figura 2.2: rossi=core-objects, gialli=border, blue=noise

verrebbe identificata un'unica componente connessa. Lasciare questi archi renderebbe l'algoritmo molto più sensibile al rumore, poiché in tal un solo punto 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
- scegliere il cluster con cui il border object ha maggiore connettività

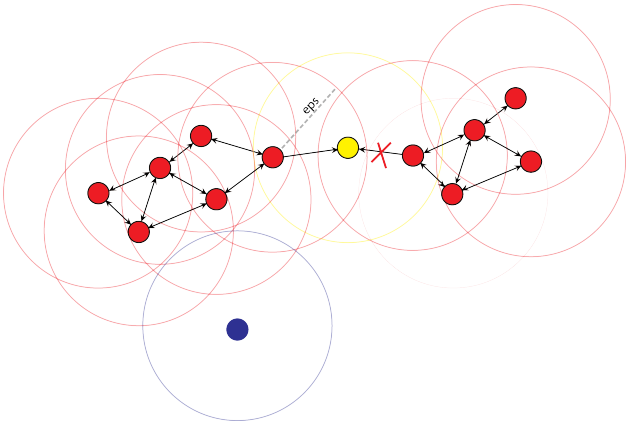


Figura 2.3: border object density reachable da due core-object

- scegliere il cluster più vicino al border object

2.2 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 [**?**] 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. Tuttavia 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) [**?**] [**?**]. 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*

2.2.1 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 $\forall (x, y) \in S, \forall h \in H$

- $if \quad d(x, y) \leq d_1 \Rightarrow P_H[h(x) == h(y)] \geq p_1,$
- $if \quad d(x, y) \geq d_2 \Rightarrow P_H[h(x) == h(y)] \leq p_2$

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 $h(q)$

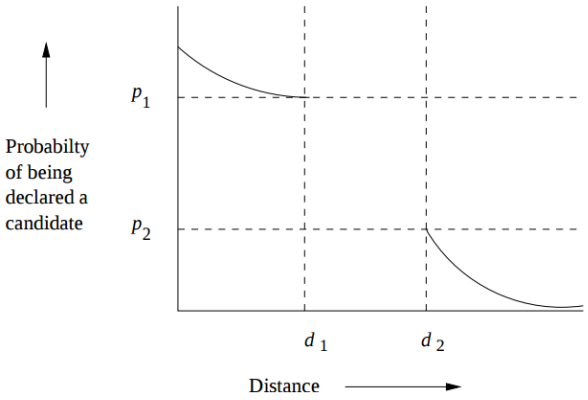
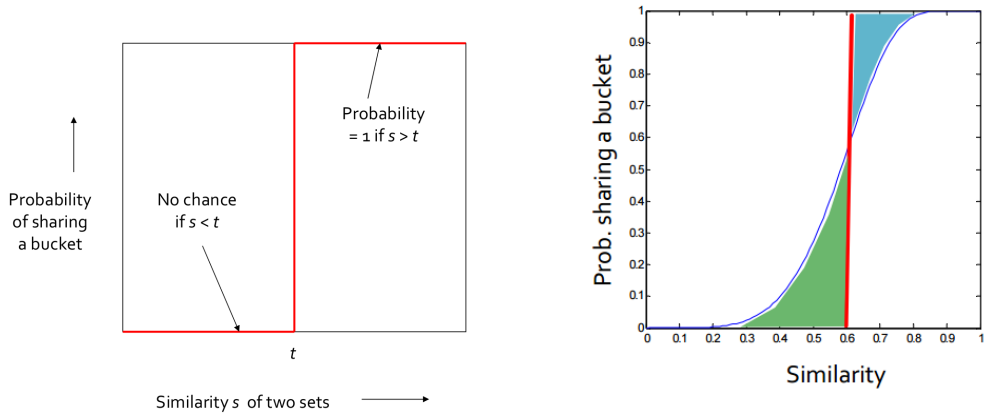


Figura 2.4: esempio di una funzione $(d_1, d_2, p_1, p_2) - sensisitve$

La figura **??** 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, allora $1-p$ è la probabilità che non lo sia. Avendo b funzioni $h_1, ..., h_b$ avre-



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

Figura 2.5: esempi di S-Curve

mo 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 **??** 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 **??** 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.

2.2.2 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[**?**] 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:

$$h_{\vec{r}}(\vec{u}) := \begin{cases} 1 & \text{se } \vec{r} \cdot \vec{u} \geq 0, \\ 0 & \text{se } \vec{r} \cdot \vec{u} \leq 0, \end{cases} \tag{2.1}$$

La figura **??** mostra l'interpretazione geometrica dell'equazione **??**, 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 maniera una funzione di hashing si avrà che, per due vettori $\vec{u}, \vec{v}$,

$$P[h_{\vec{r}}(\vec{u}) = h_{\vec{r}}(\vec{v}))] = 1 - \frac{\theta(\vec{u}, \vec{v})}{\pi} \tag{2.2}$$

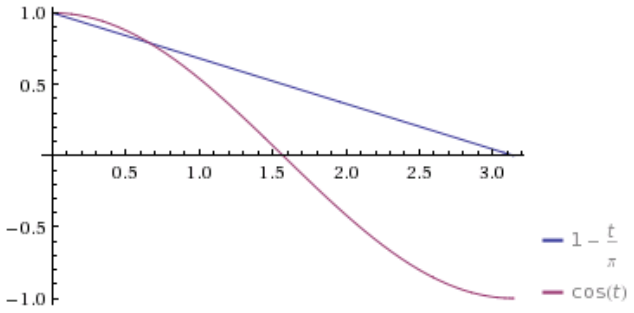


Figura 2.6: approsimazione sim. coseno

Secondo il teorema**??** [**?**], la probabilità che un iperpiano random, separi due vettori è direttamente proporzionale all'angolo fra di essi. In figura **??** 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 **??** si ha che

$$cos(\theta(u, v)) = cos((1 - P[h_r(u) = h_r(v)])\pi) \tag{2.3}$$

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.

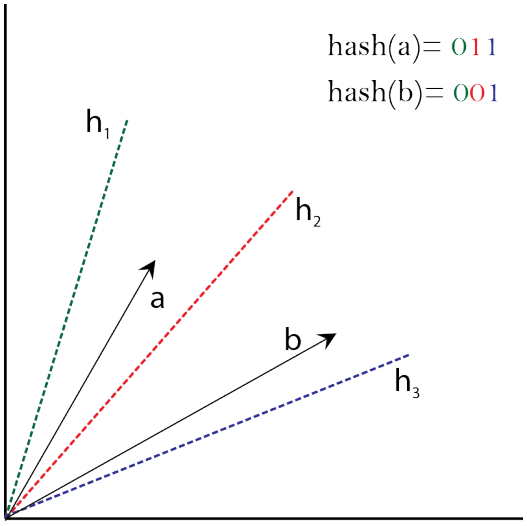


Figura 2.7: funzione di hashing