Alma Mater Studiorum · Università di Bologna

**SCUOLA DI SCIENZE**
**Corso di Laurea in Informatica Magistrale**

# Metodi per il Topic Detection su Twitter

**Relatore:**
**Chiar.mo Prof.**
**FABIO TAMBURINI**

**Presentata da:**
**FRANCESCO TROMBI**

**Sessione II**
**Anno Accademico 2016/2017**

*A Federica.*

# Indice

# Introduzione

I *social network* hanno cambiato radicalmente la quotidianità, semplificandone ed arricchendone diversi aspetti. Ma non solo. Ogni giorno sui social network vengono prodotte quantità enormi di dati, dai quali si possono ricavare numerose informazioni. Si è dunque assistito ad una spinta, causata dall'aumento esponenziale degli utilizzatori di social network, a settori dell'informatica che si occupano di sviluppare tecnologie in grado di dare un significato in maniera automatica a questi dati.

Uno di questi settori è quello dell'elaborazione del linguaggio naturale, che si occupa dello sviluppo di tecniche e strumenti per analizzare e dunque comprendere dati prodotti in linguaggio naturale. I dati non devono necessariamente provenire dai social network, ma essi rappresentano ormai una fonte di informazioni non trascurabile, dunque il numero di strumenti ad essi dedicati sta aumentando.

Uno dei task contenuti nell'elaborazione del linguaggio naturale è il task di *topic detection*, cioè il determinare gli argomenti discussi dai dati in input. Si tratta sicuramente di un task moderno ed affascinante, intorno al quale si stanno effettuando diversi studi incentrati sulla ricerca di algoritmi dalle prestazioni sempre migliori.

Quello del topic detection è dunque un contesto vivace: in esso si va a collocare questo lavoro di tesi. Verrà infatti proposto un sistema che risolva il task di topic detection, prendendo in input i dati raccolti dal social network *Twitter*.

Si descriverà il task nel dettaglio e verranno illustrate differenti tecnologie storicamente utilizzate per risolverlo. Si presenterà nel dettaglio l'implementazione della soluzione ed infine si mostreranno i risultati in output. Tale sistema è stato sviluppato per essere inserito in un software più grande, ma possiede diverse caratteristiche che ne garantiscono l'utilizzo, applicando poche modifiche, anche all'esterno.

Il primo capitolo tratterà il problema del topic detection, fornendone una panoramica ed una collocazione precisa all'interno degli ambiti che compongono l'elaborazione del linguaggio naturale. Verrà poi presentato nel dettaglio il problema che questo lavoro di tesi intende risolvere e verranno fornite le nozioni e le definizioni basilari per la comprensione delle tematiche trattate.

Nel secondo capitolo verranno descritte le tecnologie per risolvere il problema del topic detection, attraverso un percorso che le presenta in ordine cronologico. Verrà posta particolare attenzione alla descrizione degli algoritmi utilizzati durante l'implementazione della soluzione.

Il terzo capitolo si occuperà di descrivere le tecniche di clustering che sono state studiate ed utilizzate durante l'implementazione della soluzione proposta. Tali tecniche, come verrà descritto, sono centrali nello sviluppo di un sistema che intende risolvere il task di topic detection.

Nel quarto capitolo verrà presentata la struttura della soluzione implementata, con particolare attenzione alla descrizione dell'architettura e degli strumenti software studiati ed utilizzati durante lo sviluppo.

Nel quinto capitolo verrà presentata l'implementazione della soluzione, concentrandosi sui punti più importanti del codice sorgente che la compone. Verranno descritte nel dettaglio tutte le fasi che permettono di raggiungere l'output desiderato.

Nel sesto ed ultimo capitolo si descriveranno i problemi riscontrati durante la fase di valutazione del sistema. Si proporrà una soluzione ad essi ed una metodologia di valutazione del sistema. Verranno analizzati i risultati prodotti per effettuare una valutazione delle prestazioni del sistema.

Nelle conclusioni si farà un breve riassunto della descrizione del lavoro di tesi, concludendo con i possibili sviluppi futuri che possono integrare e migliorare la soluzione.

# Capitolo 1

# Topic Detection

Questo capitolo si concentra sul problema del Topic Detection, fornendone una panoramica ed una collocazione all'interno dei diversi ambiti dell'informatica, per meglio comprendere il problema affrontato dal lavoro di tesi e la soluzione proposta.

Verrà dunque descritto il Topic Detection ed il contesto in cui si colloca, e verrà data una serie di definizioni che permetteranno di comprendere le nozioni elementari ad esso collegate; infine verrà presentato il problema affrontato da questo lavoro di tesi.

## 1.1 Topic Detection and Tracking

Il Topic Detection è uno dei tanti *task* presenti nel problema del *Topic Detection and Tracking* (da qui in poi abbreviato in TDT). Una buona introduzione all'argomento si trova nel primo capitolo del libro *Topic detection and tracking*, scritto da Fiscus e Doddington [1]. L'obiettivo del TDT è quello di sviluppare tecnologie che cerchino, organizzino e diano una struttura a materiali testuali in diverse lingue.

Il TDT si colloca all'interno dell'elaborazione del linguaggio naturale, una branca dell'informatica concentrata sulle tecniche necessarie ad elaborare tramite un calcolatore elementi testuali ed audio prodotti in linguaggio naturale. I task dell'elaborazione del linguaggio naturale spaziano dall'analisi sintattica all'analisi semantica e sono da sempre collegati agli obiettivi dell'intelligenza artificiale: entrambe affrontano infatti come obiettivo comune la recente sfida di implementare la comprensione automatica del linguaggio naturale.

I task dell'elaborazione del linguaggio naturale vengono spesso utilizzati in ambiti di *Information Retrieval* e *Business Intelligence* (discipline introdotte in seguito), in modalità che verranno descritte successivamente.

### 1.1.1 Social media

Vista la loro recente diffusione [2], è necessario porre un occhio di riguardo nei confronti dei *social media*, che possono essere confrontati ai media tradizionali (radio, televisione, giornali, ...) considerando diversi parametri:

- bacino d'utenza: entrambe le tipologie di media permettono un raggiungimento globale dell'utenza;
- accessibilità: i mezzi di produzione necessari ai media tradizionali sono solitamente fuori portata per il privato, mentre i costi della trasmissione tramite social media sono contenuti se non gratuiti;
- fruibilità: non è necessario essere formati per produrre contenuti per i social media, mentre per i media tradizionali vengono ancora richieste competenze specifiche;
- velocità: intesa come velocità di trasmissione, decisamente più lenta per i media tradizionali rispetto ai social media, i quali vantano come maggior pregio l'immediatezza;
- permanenza: errori commessi nei contenuti dei media tradizionali sono difficilmente rimovibili, mentre quando si parla di social media esistono mezzi quali i commenti o le modifiche per correggere errori di battitura o modificare anche radicalmente il contenuto.

Il sito internet *We Are Social* [3] pubblica ogni anno un report sulle statistiche legate all'utilizzo dei *social network* a livello mondiale, una delle categorie di social media più prolifiche di contenuti, che stima circa 2'789 milioni di utenti di social media attivi, con un incremento del 21% rispetto al gennaio del 2016 [4].

Questi ed altri motivi hanno dunque permesso ai social media di divenire pervasivi nella nostra società, e l'informatica si lega sempre di più ad essi come fonte ormai non trascurabile, se non primaria, per ricavare immense moli di dati, che spaziano dalle informazioni sull'attualità alle opinioni degli utenti.

Come vedremo successivamente, l'utilizzo dei social media può cambiare anche radicalmente diversi concetti e definizioni sviluppati nei primi anni del TDT.

### 1.1.2 Lo studio pilota

Il TDT nasce in un contesto precedente a quello dei social media appena descritto: viene infatti presentata nel 1998, con la pubblicazione, avvenuta un anno dopo, dello

studio pilota [5] promosso dalla *Defense Advanced Research Projects Agency* (DARPA) [6], l'agenzia del Dipartimento della Difesa statunitense responsabile allo sviluppo di tecnologie emergenti per scopi militari. Tale studio fu fatto per investigare sullo stato dell'arte delle tecnologie che ricercano e seguono eventi all'interno di un flusso di notizie.

Lo studio pilota ha aperto la strada ad una serie di tecnologie innovative, alcune superate, altre ancora in uso, ma non solo. Ha fornito infatti:

- una prima definizione ai termini utilizzati nel TDT;
- una serie di direttive per la valutazione dei sistemi;
- l'elenco dei task legati a tale problema.

Allo studio pilota sono seguiti riunioni annuali sull'argomento, atte alla ridefinizione della terminologia e dei task, ed all'aggiornamento delle tecnologie utilizzate per risolvere tali problemi.

### 1.1.3 Definizioni

Come accennato, anche le definizioni fornite dallo studio pilota hanno subito diverse modifiche, dovute ad aggiornamenti annuali [1].

Un *evento* è un "qualcosa che accade in un preciso momento", per darne una definizione preliminare. Volendo fare un esempio, *l'elezione del Presidente Trump* è un evento, mentre *l'elezione del Presidente* è una classe di eventi.

Un'*attività* è un insieme di eventi strettamente correlati, che accadono in ben determinati momenti o luoghi.

Una *storia* è un insieme coerente di notizie, che include due o più proposizioni indipendenti dichiarative su un singolo evento.

La definizione di *topic* è stata modificata diverse volte dopo la pubblicazione dello studio pilota, che si limitava a far combaciare il concetto di evento con quello di topic. Successivamente è stata ampliata, includendo altre attività od eventi correlati direttamente ad esso. Un *topic* è dunque un evento od un'attività che dà inizio a tutti gli eventi od alle attività direttamente correlati.

Questa definizione, pur essendo maggiormente inclusiva rispetto a quella data dallo studio pilota, non deve essere confusa con il concetto (ancora più inclusivo) che normalmente si associa alla parola italiana "argomento", traduzione letterale di topic. Per questo motivo da qui in poi verrà utilizzato il lemma inglese "topic".

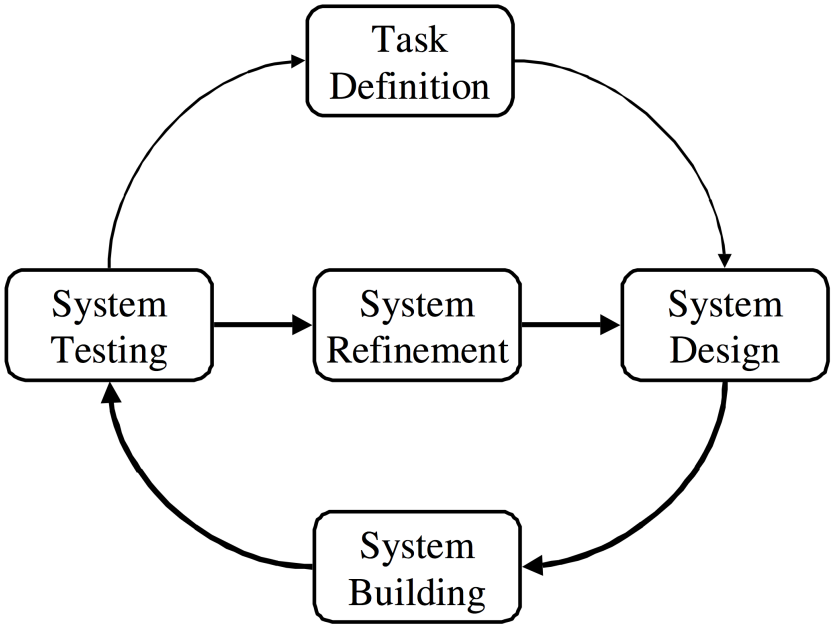


Figura 1.1: Technology Evaluation Cycle

Un *corpus* è un insieme di materiale scritto o parlato memorizzato in un calcolatore. Per svolgere i task descritti in seguito è necessario possedere corpora di grandi dimensioni.

Una storia viene considerata *in-topic* quando discute di eventi ed attività direttamente connessi all'evento che ha dato origine al topic. La nozione di "connessione" tra storie ed evento originario del topic qui utilizzata non è stata formalizzata, dunque non si hanno confini precisi per stabilire se una storia è in-topic o meno. Il *Linguistic Data Consortium* (LDC), consorzio per la distribuzione di risorse linguistiche che si occupò della trascrizione di ore e ore di discorsi tv e radio per creare il corpus dello studio pilota, ha definito però una serie di linee guida per l'annotazione manuale dei topic [7].

### 1.1.4 Metodologia di valutazione dei sistemi TDT

Poiché la definizione del problema del TDT è stato storicamente introdotto dallo studio pilota della DARPA, al centro della metodologia di valutazione si trova il *technology evaluation cycle* (mostrato nella figura 1.1) utilizzato proprio dal reparto di ricerca e sviluppo dei programmi in ambito dell'elaborazione del linguaggio. Il ciclo ha principalmente cinque fasi:

1. definizione del task;

2. progettazione del sistema;
3. implementazione del sistema;
4. testing del sistema;
5. raffinamento del sistema.

Conclusa la fase di raffinamento, il sistema viene testato nuovamente per valutare se tali cambiamenti hanno migliorato le prestazioni del sistema nelle modalità previste.

### TDT task valutati come *detection task*

Tutti i task vengono considerati come *detection task*, cioè task in cui vengono presentati al sistema i dati di input e le ipotesi su di essi, ed esso valuta se tali ipotesi sono vere o no. Anche queste nozioni vengono descritte da Fiscus e Doddington [1].

La procedura di decisione della validità delle ipotesi viene chiamata *trial*: se le ipotesi sono vere, il trial viene chiamato *target trial*, altrimenti *not-target trial*. Naturalmente il sistema può correttamente individuare un target, oppure non individuarlo: in quest'ultimo caso si parla di *missed detection*. Un not-target può essere etichettato allo stesso modo in maniera corretta, oppure può essere erroneamente etichettato come target. In quest'ultimo caso si parla di *false alarm*.

Al di là dell'effettiva decisione, il sistema emette un *decision score*, o punteggio di decisione, che rappresenta la forza della convinzione del sistema nel definire il trial come target. Ogni sistema può implementare il proprio spazio di punteggi, ma essi devono essere confrontabili tra diversi topic e corpus (la soluzione più comune è normalizzarli opportunamente).

Esistono due tecniche principali per rappresentare le performance in termini di missed detection e false alarm:

1. *detection cost function ($C_{Det}$)*: valore singolo per misurare le prestazioni del sistema in base alla decisione di etichettare il trial come target o not-target;
2. *decision error tradeoff curve (curve DET)*: visualizzazione del tradeoff tra missed detection e false alarm usando la distribuzione dei punteggi di decisione.

Poiché nel processo di valutazione di un sistema TDT vengono utilizzati diversi topic, è buona norma fare una media dei valori $C_{Det}$ e delle curve DET tra tutti i topic. Tali misure della performance basate sulla media vengono dette metriche *topic-weighted*. Il vantaggio principale nell'utilizzo di queste metriche è che permettono di stabilire con

facilità gli intervalli di confidenza (intervalli in cui sono contenuti i valori plausibili per i parametri) e dunque identificare in maniera altrettanto semplice i valori anormali.

In alternativa alle metriche topic-weighted, le prestazioni globali potrebbero essere valutate usando o $C_{Det}$ o curve DET *trial-weighted* (dette anche *story-weighted*, dato che i trial sono decisioni generalmente basate su storie). Lo svantaggio è che topic con un alto numero di trial possono oscurare i valori di topic dalle dimensioni inferiori.

### *Detection Cost Function* Normalizzata

La performance dei sistemi di detection viene definita in termini di probabilità di errori, cioè la probabilità di missed detection $P_{Miss}$ e probabilità di false alarm $P_{Fa}$. Tali probabilità vengono combinate linearmente in un valore singolo, il $C_{Det}$, assegnando un costo costante ad entrambe le tipologie di errore e specificando la probabilità di un target a priori. Il modello basato su $C_{Det}$ fornisce un'interfaccia opportuna per valutare il tradeoff di performance tra $P_{Miss}$ e $P_{Fa}$.

Tipicamente, un utente utilizza una tecnologia che permette di filtrare dei contenuti (compito che svolgono anche le tecnologie TDT) per ridurre il carico di lavoro: per esempio, se si vogliono cercare tutte le storie collegate ad un evento senza leggere milioni di storie è necessario filtrarle in base al topic. Per un utente di questo tipo esistono due costi: uno dato dalla lettura della storia, ed uno dato dalla lettura di una storia etichettata in maniera scorretta, che considera la perdita di tempo.

Ragionando in questo modo $C_{Det}$ si basa su $C_{Miss}$ e $C_{Fa}$ per stimare questi due costi: una combinazione lineare di $P_{Miss}$ e $P_{Fa}$ utilizzando i costi assegnati sarebbe sufficiente se il valore di target e not-target fosse identico. Nei sistemi TDT e nella maggior parte dei sistemi di filtering non è però così: la differenza tra i due si misura in ordini di grandezza, perciò per compensare la differenza del valore dei target viene introdotto un altro termine, $P_{Target}$. La formula per esprimere $C_{Det}$ diventa dunque:

$$C_{Det} = (C_{Miss} * P_{Miss} * P_{Target} + C_{Fa} * P_{Fa} * (1–P_{Target})) \tag{1.1}$$

con:

$$P_{Miss} = \#MissedDetection/\#Target \tag{1.2}$$

$$P_{Fa} = \#FalseAlarm/\#NotTarget \tag{1.3}$$

Dove:

- $C_{Miss}$ e $C_{Fa}$ sono rispettivamente i costi delle missed detection e dei false alarm, e sono dati al sistema in base all'applicazione;

- $P_{Miss}$ and $P_{Fa}$ sono rispettivamente le probabilità di una missed detection e di un false alarm, e sono il risultato della valutazione del sistema;
- $P_{Target}$ è la probabilità calcolata a priori di trovare un target come specificato dall'applicazione.

Per ogni task TDT, le specifiche della valutazione dichiarano i valori $C_{Miss}$ e $C_{Fa}$, impostati a seconda delle precedenti esperienze con lo sviluppo di sistemi di detection: per la maggior parte dei task di valutazione dei sistemi TDT vengono rispettivamente impostati a 10 e 1. $P_{Target}$ è invece basato su calcoli statistici sul corpus e misura la ricchezza delle storie on-topic del training set. Si noti come questi valori siano scelti in maniera arbitraria: l'importante è che, se utilizzati come costanti, il loro valore è meno importante del loro utilizzo.

Nonostante $C_{Det}$ sia una buona metrica per valutare le performance, l'intervallo dinamico di valori rende difficile interpretare i risultati. Perciò nel TDT è necessario introdurre la versione normalizzata della Detection Cost Function, cioè $(C_{Det})_{Norm}$. L'obiettivo della normalizzazione è quello di trasporre la finestra di valori in un intervallo più semplice da interpretare: per fare ciò, $C_{Det}$ viene diviso per il costo minimo supposto di un sistema che ad ogni domanda risponde o sempre "sì" o sempre "no". In questo modo il valore minimo è sempre 0, ma un valore di 1.0 indica che non cambia nulla dal rispondere sistematicamente con la stessa risposta a tutte le domande. La formula di $(C_{Det})_{Norm}$ diventa:

$$(C_{Det})_{Norm} = C_{Det}/MIN((C_{Miss} * 1.0 * P_{Target} + C_{Fa} * 0.0 * (1\text{–}P_{Target})), \\ (C_{Miss} * 0.0 * P_{Target} + C_{Fa} * 1.0 * (1\text{–}P_{Target}))) \quad (1.4)$$

che, semplificata, si riduce a:

$$(C_{Det})_{Norm} = C_{Det}/MIN((C_{Miss} * P_{Target}), (C_{Fa} * (1\text{–}P_{Target})) \quad (1.5)$$

### Curve DET

Le curve DET sono visualizzazioni del tradeoff tra i $P_{Miss}$ e $P_{Fa}$. Le curve sono costruite stimando e disegnando $P_{Miss}$ e $P_{Fa}$ per ogni punto dello spazio dei punteggi di decisione, in un grafico dove l'asse delle $x$ rappresenta la probabilità in percentuale di false alarm, mentre l'asse delle $y$ rappresenta la probabilità in percentuale di missed detection. Poiché queste due probabilità sono legate ad errori, buone performance saranno mostrate da curve posizionate verso l'angolo basso sinistro (cioè con basse probabilità per entrambe le tipologie di errore).

Anche in questa metrica, come in $C_{Det}$, è preferibile utilizzare il paradigma topic-weighted, perché utilizzando quello story-weighted si riscontrano gli stessi problemi mostrati in precedenza.

### Precision, Recall e F-Score

Come verrà esposto nel capitolo successivo, il problema del TDT può essere collocato nel contesto dell'*Information Retrieval*. Momentaneamente senza entrare nel dettaglio, è possibile ereditare da questa disciplina le tecniche di valutazione dei sistemi, che utilizzano un approccio differente rispetto a quelli basati su missed detection e false alarm. Le metriche introdotte da Baeza-Yates e Ribeiro-Neto nel loro libro *Modern Information Retrieval* [8] sono principalmente due: *precision* e *recall*.

Si consideri dunque un sistema che deve svolgere un task di detection. Esso necessita di un'informazione $I$ per individuare i documenti rilevanti presenti all'interno di un corpus. Si pongano come insiemi di riferimento:

- $R$, l'insieme dei documenti ritenuti rilevanti dal sistema in base all'informazione $I$;
- $A$, l'insieme dei documenti realmente rilevanti, non conosciuto dal sistema ma da chi deve valutarlo, che rappresenta dunque l'insieme di documenti che il sistema avrebbe dovuto ritenere rilevanti;
- $R_A$, ottenuto dall'intersezione tra $R$ ed $A$, che rappresenta dunque i documenti correttamente individuati dal sistema.

Viene chiamata *precision* la metrica che descrive la frazione di documenti individuati dal sistema che sono effettivamente rilevanti, ed è descritta dalla formula 1.6.

$$precision = \frac{|R_A|}{|R|} \tag{1.6}$$

In altre parole la precision è il rapporto tra il numero di documenti che il sistema ha correttamente individuato come rilevanti ed il numero di documenti considerati rilevanti dal sistema.

Viene chiamata *recall* la metrica che descrive la frazione dei documenti rilevanti che sono stati correttamente rilevati, ed è descritta dalla formula 1.7.

$$recall = \frac{|R_A|}{|A|} \tag{1.7}$$

In altre parole la recall è il rapporto tra il numero di documenti che il sistema ha correttamente individuato come rilevanti ed il numero di documenti effettivamente rilevanti.

Esiste infine una metrica, chiamata *F-score*, che combina le due metriche precedenti, calcolandone la media armonica e, dunque, restituendo un risultato numerico compreso tra 0 (il risultato peggiore) ed 1 (perfezione). Il valore di F-score viene calcolato con la formula 1.8.

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{1.8}$$

### 1.1.5 I task del TDT

I task del TDT introdotti dallo studio pilota [5], come le definizioni date in precedenza, hanno subito diverse modifiche nelle riunioni successive allo studio pilota [1]. I task del TDT, poi descritti nel dettaglio, sono i seguenti:

- *topic tracking*;
- *link detection*;
- *topic detection*;
- *first story detection*;
- *story segmentation.*

#### Topic Tracking

I sistemi di topic tracking individuano storie che trattano di un topic conosciuto a priori dal sistema. Al sistema viene dunque dato un topic, ed ogni storia successiva deve essere etichettata in base alla sua appartenenza o meno a tale topic. Essendo il topic definito come l'insieme delle storie correlate direttamente ad esso, al sistema di tracking serve un corpus diviso in due parti: *training set* e *test set*. Ogni storia del training set viene etichettata in base all'appartenenza al topic: è dunque un insieme di storie on-topic. Le etichette a tali storie, insieme al testo delle storie, sono le uniche informazioni che il sistema possiede per definire il topic e per poter correttamente classificare le storie contenute nel test set.

Un parametro delicato da definire per lo svolgimento del topic tracking task è $N_t$, cioè il numero di storie utilizzate per definire il topic, dunque la dimensione del training set. Il test set viene di conseguenza ricavato dalle storie rimanenti. I sistemi vengono valutati in base alla loro capacità di definire le storie rimanenti come on-topic ed off-topic.

Gli sviluppatori devono considerare però tre problematiche durante la fase di progettazione del sistema. Il primo problema è dovuto al fatto che i sistemi di topic tracking devono essere addestrati e testati su ogni topic in maniera indipendente. I sistemi non possono utilizzare in nessun modo le definizioni di un altro topic (dunque le storie ad esso correlate), anche se questo renderebbe più semplice il completamento del task. Considerando tale indipendenza, i parametri del modello (come le iterazioni del training, la dimensione dei set di training e test, ...) devono essere cambiati per ogni topic, cercando i valori più adatti: dato che il numero di storie cambia da topic a topic è preferibile

utilizzare metriche topic-weighted. L'indipendenza dei topic porta però ad avere un vantaggio: è possibile valutare separatamente storie che trattano più topic, a patto che tale molteplicità venga opportunamente gestita al momento di comporre i risultati.

Il secondo problema è decidere come normalizzare il punteggio di decisione attraverso i vari topic perché, come detto in precedenza, devono essere confrontabili tra loro. Si noti come questo problema sarebbe più semplice senza l'indipendenza dei topic.

Il terzo problema è dovuto alla necessità del sistema di essere multi-lingua.

### Link Detection

Un sistema per risolvere il link detection task è in grado di capire se due storie trattano lo stesso topic. Il sistema dunque risponde alla domanda "queste due storie appartengono allo stesso topic?", semplicemente con "sì" o "no".

Come per altri task, il sistema allega all'output un punteggio di decisione se la risposta è affermativa. La decisione ed il suo eventuale punteggio vengono usate per calcolare la $C_{Det}$ e le curve DET.

Questo task può essere visto come una funzionalità chiave dalla quale vengono costruiti i sistemi di topic tracking (e, come vedremo successivamente, di topic detection): è infatti connesso al topic tracking, perché serve per determinare se la storia elaborata appartiene o no al topic considerato. La differenza è che preferisce sotto-campionare lo spazio delle storie per prendere una decisione, in modo da essere più efficiente. Tale paradigma consente di avere diversi vantaggi, tra i quali spiccano:

- le performance possono essere valutate da un giudice umano senza dare una definizione formale di topic;
- poiché lo spazio dei topic non deve essere diviso in cluster di storie, la gestione di storie che parlano di più topic è un non problema.

### Topic Detection

Il topic detection task valuta tecnologie in grado di individuare topic non conosciuti dal sistema. Come nel topic tracking task un topic è definito dall'insieme di storie ad esso collegate; ma in questo caso non si hanno conoscenze a priori su nessuno dei topic. Il sistema deve dunque implementare una metodologia per definire ciò che costituisce un topic, in maniera indipendente rispetto ai topic specifici.

Il sistema deve individuare dei *cluster*, cioè gruppi di elementi omogenei in un insieme di dati, composti di storie che trattano dello stesso topic. Il concetto di clustering viene

facilmente applicato a storie che trattano di notizie, ma la valutazione delle performance è difficile, perché spesso le storie sono correlate a più topic. Questo fenomeno non solo rende l'elaborazione di una storia dipendente dalle storie elaborate in precedenza, ma rende anche lo studio delle performance su insiemi casuali di storie fuorviante.

La valutazione deve tenere conto inoltre dell'indipendenza dei topic: perciò le storie correlate a più topic vengono dichiarate non classificabili, anche se il sistema effettua un clustering su tutte le storie: in questo modo le storie di questa tipologia contribuiscono al sistema, senza però contribuire alle misure legate agli errori. La valutazione delle performance per il topic detection utilizza metriche topic-weighted come $C_{Det}$ o le curve DET, oppure precision e recall.

Tale task prevede due casistiche: la ricerca dei topic può avvenire in maniera retrospettiva su una collezione di dati, oppure on-line elaborando un flusso di dati in tempo reale. Entrambe assumono in partenza che ogni storia tratti di uno ed uno solo argomento.

1. *retrospective topic detection*: i topic vengono tutti identificati dentro ad un corpus di storie; ogni topic è definito in base alla sua associazione con le storie, perciò il task consiste nel raggruppare tali storie in *cluster*, ed ogni cluster rappresenta un topic e le storie che lo compongono trattano dell'evento originario del topic. Per ciò che si è assunto, ogni storia verrà collocata in uno ed un solo cluster;

2. *on-line new topic detection*: l'obiettivo è identificare nuovi eventi in un flusso di storie. Ogni storia viene elaborata in sequenza ed è necessario che il sistema decida se essa tratta di un nuovo argomento. Tale decisione viene presa dopo che la storia in oggetto è stata elaborata, ma prima che venga elaborata la storia successiva.

### First Story Detection

Il task di first story detection valuta tecnologie in grado di individuare la prima storia che tratta un topic. Questo task può essere visto un caso del topic detection, che si concentra su aspetti connessi con la *novel information detection*, cioè per esempio quando iniziare a definire un nuovo cluster. L'input al sistema è identico a quello dei sistemi di topic detection: la vera differenza è nell'output.

I sistemi di first story detection producono in output una risposta secca che risponde alla domanda "questa storia tratta di un nuovo topic?", insieme al solito punteggio di decisione se la risposta è affermativa.

**Story Segmentation**

Il task di story segmentation prevede la suddivisione del testo nelle sue storie. I sistemi di story segmentation valutano tecnologie in grado di comprendere quando una storia cambia, trasformando flussi di testo in storie formattate per i task TDT.

Riprendendo la definizione di storia vista in precedenza, cioè che "insieme coerente di notizie, che include due o più proposizioni indipendenti dichiarative su un singolo evento", si può notare come sia stata data in modo da escludere gli slogan pubblicitari. Questo perché storicamente i corpus per il TDT venivano ricavati da trascrizioni automatiche di discorsi parlati presi da radio e televisioni (media tradizionali), dunque un contesto in cui la pubblicità avrebbe solamente "sporcato" le analisi. Come vedremo successivamente, tale problema non si pone più con l'avvento dei social media, e la definizione di storia andrà rivista.

Il task di story segmentation viene visto come necessario per iniziare tutti gli altri task: effettivamente senza una corretta suddivisione in storie nessuno degli altri task TDT può iniziare. Anche questo task deve essere multi-lingua.

La valutazione delle performance si basa sempre sulle metriche presentate in precedenza, ma il calcolo di $P_{Miss}$ e $P_{Fa}$ sono differenti: la performance del sistema viene infatti giudicata in base a quanto i confini della storia coincidano con quelli di riferimento.

## 1.2 Definizione del problema

Ora che sono state definite le linee guida per comprendere la strutturazione di un sistema di topic detection, è possibile introdurre il problema affrontato in questo lavoro di tesi.

Il punto di partenza è un software *open source* sviluppato per la *Business Intelligence*. La business intelligence è un termine che raccoglie in sé diverse definizioni, non esclusive:

- insieme di processi aziendali per raccogliere dati ed analizzare informazioni strategiche;
- la tecnologia utilizzata per realizzare questi processi;
- le informazioni ottenute come risultato di questi processi.

Il software di partenza è la suite SpagoBI [9], offerta in versione "*community edition*" dall'azienda Engineering in maniera gratuita ed open source. Oltre ai diversi moduli legati all'analisi dei dati e dei processi aziendali, SpagoBI include una componente di

*social network analysis* (metodologia di analisi delle relazioni sociali, applicata in questo caso ai social network), nello specifico collegata a Twitter, per effettuare analisi di topic detection e *sentiment analysis*.

Il sentiment analysis, chiamata anche *opinion mining*, è un campo di studi che analizza le opinioni, i sentimenti, le valutazioni, le stime, le attitudini e le emozioni delle persone nei confronti di prodotti, servizi, organizzazioni, individui, problemi, eventi e topic, e delle loro caratteristiche, come definito da Liu [10].

Il punto di partenza è stato dunque quello di analizzare il sistema implementato da SpagoBI nelle componenti che si occupano di topic detection, per migliorarlo sostituendo la parte di topic detection con una versione aggiornata e che implementasse algoritmi dalle performance migliori. Nonostante le fasi di configurazione della suite si siano rivelate complesse a causa della documentazione, scarsa e spesso errata, il funzionamento del topic detection in SpagoBI è semplice da spiegare:

- nel momento in cui si lancia il modulo di social network analysis il software inizia a scaricare dei tweet;
- si attende la conclusione del download;
- vengono lanciati due script scritti in linguaggio R, che si occupano rispettivamente del sentiment analysis e del topic detection;
- i risultati degli script vengono inseriti in un database relazionale e mostrati a schermo attraverso un'interfaccia grafica.

Durante la configurazione della suite si è scelto di utilizzare come DBMS MySQL, a causa di diversi problemi di compatibilità riscontrati tramite l'utilizzo di altri sistemi.

Come accennato, per questo lavoro di tesi si è deciso di procedere sostituendo lo script R nativo per il topic detection con un nuovo script. Per implementare il nuovo script si è scelto di utilizzare il linguaggio Python, nella versione 3.6, grazie alle numerose librerie sviluppate per risolvere i problemi legati al topic detection.

Lo script R nativo è stato mantenuto solamente come interfaccia per lanciare tale script Python, in modo da rendere quest'ultimo un semplice modulo del sistema, facilmente estraibile per utilizzi all'interno di altri sistemi. Tale script si interfaccia al database di SpagoBI in maniera analoga rispetto allo script nativo. I dettagli implementativi di tale codice verranno descritti successivamente, nel capitolo dedicato all'implementazione della soluzione.

Il lavoro di tesi si concentra dunque sul miglioramento delle prestazioni di un sistema di topic detection retrospettivo utilizzato su tweet all'interno di una suite software di business intelligence.

### Variazioni rispetto al TDT classico

Prima di descrivere le diverse tecnologie candidate alla risoluzione del problema posto al base di questo lavoro di tesi, è opportuno elencare alcune variazioni che vengono effettuate rispetto a quanto detto in questo capitolo.

Per prima cosa non è necessario effettuare alcuna segmentazione sui dati ricavati da Twitter, poiché essi arrivano già segmentati in tweet. Di conseguenza, non è possibile accettare come definizione di storia quella data dallo studio pilota e dai meeting successivi ad esso: si ricorda infatti che tale definizione pone una storia come "insieme coerente di notizie, che include due o più proposizioni indipendenti dichiarative su un singolo evento". Si capisce come tale definizione non possa calzare utilizzando come unità di base del lavoro i tweet, spesso composti da una singola proposizione dichiarativa. È possibile effettuare tale passaggio per via della motivazione che aveva portato ad escludere frasi composte da una sola proposizione: come spiegato nella sezione precedente, era dovuto alla volontà di eliminare dal corpus gli slogan pubblicitari. In un social media, dove la maggior parte dei dati viene prodotta da utenti, tale problematica non sussiste più, dunque è stato necessario aggiornare tale definizione. Per quanto riguarda questo lavoro di tesi, una storia coincide con un tweet.

Questa scelta porta però alla nascita di un problema. Come detto in precedenza il topic detection è un task non supervisionato: il sistema non necessita dunque di un corpus annotato per etichettare le storie, ma deve essere utilizzato nella fase di valutazione del sistema. In questo lavoro di tesi, estremamente specifico nel corpus e negli obiettivi, non è stato possibile trovare un corpus annotato di tweet in italiano per testare il sistema. Per questo motivo la tipologia di valutazione è stata qualitativa più che formale, ma è necessario sottolineare l'importanza di sviluppi futuri legati alla creazione di un corpus opportuno per la valutazione del sistema proposto.

# Capitolo 2

# Tecnologie per il Topic Detection

In questo capitolo il lavoro di tesi si incentra su uno specifico task del TDT, cioè il topic detection. Per poter comprendere l'approccio al problema presentato da questo lavoro di tesi e le scelte effettuate durante l'implementazione della soluzione, è necessario descrivere un percorso legato alle tecnologie per il topic detection storicamente utilizzate e quelle innovative.

Vengono di seguito definiti alcuni concetti preliminari necessari per comprendere meglio lo sviluppo delle tecnologie legate al topic detection.

Una *parola* (o termine) è l'unità di base dei dati discreti, per definizione inserita all'interno di un dizionario con indici $\{1, \ldots, V\}$. Spesso le parole sono rappresentate come segue: la parola $w$ è un vettore tale che $w_v = 1$ e $w_u = 0$, con $v$ indice della parola nel vocabolario e $u \neq v$.

Un *documento* è una sequenza di $N$ parole. Il documento $i$-esimo viene denotato come come $d_i = w_1, w_2, \ldots, w_N$, dove $w_j$ è la $j$-esima parola del documento.

Un *corpus* è una collezione di $M$ documenti, denotato da $D = \{d_1, d_2, \ldots, d_M\}$.

## 2.1 *Vector Space Model* e riduzione dimensionale

Le maggiori spinte alle tecnologie utilizzate per il topic detection (ma in generale per tutte le discipline legate all'elaborazione del linguaggio naturale) derivano dalle ricerche fatte nel campo dell'*information retrieval* (IR).

Un libro fondamentale per la comprensione dell'information retrieval è stato scritto da Baeza-Yates e Ribeiro-Neto [11]. Gli autori definisco l'information retrieval come un campo dell'informatica che si occupa della rappresentazione, dell'immagazzinamento,

dell'organizzazione e dell'accesso di dati, intesi come oggetti contenenti informazioni. La rappresentazione e l'organizzazione di tali elementi devono fornire all'utente una modalità semplice di accesso alle informazioni a cui è interessato.

Più o meno nello stesso periodo (l'anno è sempre il 1999) anche il libro di Manning e Schütze si occupa dell'information retrieval, affermando che esso riguarda lo sviluppo di algoritmi e modelli per il recupero di informazioni da insiemi di documenti [12]. Per Manning e Schütze l'information retrieval è strettamente connessa alla disciplina dell'elaborazione del linguaggio naturale: la coincidenza di obiettivi degli studiosi di entrambi i campi ha portato allo sviluppo delle tecniche illustrate nel seguito del capitolo.

Per definire in maniera più precisa l'information retrieval è necessario contrapporla al *data retrieval* [13], che si occupa di ricercare i documenti in base ad una chiave di ricerca fornita dall'utente, che spesso però non è sufficiente a soddisfare la richiesta dell'utente stesso. L'utente di un sistema information retrieval è infatti maggiormente interessato ad ottenere informazioni riguardo ad un soggetto piuttosto che dati che soddisfano una query.

La differenza principale consiste però nel fatto che l'information retrieval ha a che fare con testi in linguaggio naturale, dunque non sempre ben strutturati e potenzialmente ambigui nella semantica. Un sistema di data retrieval (come, per esempio, un database relazionale) ha a che fare con dati dalla struttura e dalla semantica ben definite.

Il punto di partenza, come descritto nell'articolo di Lenci [14], è considerare la semantica dei documenti come *semantica distribuzionale*. Per semantica distribuzionale si intende una semantica basata sull'*ipotesi di distribuzionalità*, così definita: il grado di similarità semantica tra due espressioni linguistiche $A$ e $B$ è funzione della similarità dei contesti linguistici in cui $A$ e $B$ possono apparire.

Perciò almeno alcuni aspetti del significato di un'espressione lessicale dipendono dalle proprietà distribuzionali di tale espressione, ossia dai contesti linguistici in cui viene osservata.

Prendendo la descrizione di Arguello [15], si trova un punto di connessione tra la semantica distribuzionale e l'information retrieval nella definizione del task di ricerca: data una query ed un corpus, si cerca di trovare gli elementi *rilevanti*. Per *rilevanza* si intende la soddisfazione del bisogno di informazioni dell'utente. Un modello di information retrieval è un metodo formale che predice il grado di rilevanza di un documento rispetto ad una query.

Il modello che ha avuto maggior successo, in diversi ambiti, è stato il *Vector Space Model*, introdotto da Salton et al. [16]. Il concetto alla base del vector space model per i corpora testuali è quello di trasformare i documenti del corpus in vettori di numeri reali. Questo metodo si basa su due concetti fondamentali:

Tabella 2.1: Rappresentazione binaria dei documenti

| | $w_1$ | $w_2$ | ... | $w_m$ |
|---|---|---|---|---|
| $d_1$ | 1 | 1 | ... | 1 |
| $d_2$ | 1 | 0 | ... | 0 |
| ... | ... | ... | ... | ... |
| $d_n$ | 1 | 0 | ... | 1 |

- *metafora geometrica*: i documenti testuali vengono trasformati in oggetti geometrici, con tutte le conseguenze del caso. Si pensi solamente a come questa metafora permetta di manipolare tali documenti come oggetti geometrici, applicando loro, per esempio, algoritmi di clustering o calcoli sulla distanza (descritti successivamente);
- ipotesi di distribuzionalità, che permette di affermare, in correlazione alla metafora geometrica, che due vettori vicini corrispondono ad espressioni lessicali simili.

Un'idea di modello basilare per la trasformazione di documenti in vettori è quello della rappresentazione binaria di un documento. Dati:

- un corpus (insieme di documenti) $D = \{d_1, d_2, \ldots, d_n\}$;
- un dizionario (insieme di parole univoche) $W = \{w_1, w_2, \ldots, w_m\}$;
- una matrice $A$ di dimensione $n \times m$;

si riempiono le celle della matrice come segue:

- $A_{i,j} = 1$ se la parola $w_j$ compare nel documento $d_i$;
- $A_{i,j} = 0$ se la parola $w_j$ non compare nel documento $d_i$.

Si rappresenta dunque ogni porzione di testo come un vettore nello spazio $m$-dimensionale. Un esempio di tale rappresentazione viene mostrato nella matrice 2.1.

Al momento della query, del task di ricerca, il vector space model crea una classifica dei vettori basata sulla *similarità* tra il vettore della query $\vec{x}$ e quello del documento $\vec{y}$. Il problema si sposta ora sulla metrica per misurare tale similarità.

Un primo modo può essere quello di contare il numero di parole in comune tra i due vettori, dunque effettuando il prodotto scalare $\vec{x} \cdot \vec{y}$. Tale metrica necessità però di essere normalizzata, se si desidera utilizzarla per confrontare due sistemi.

Una metrica maggiormente utilizzata, anche grazie al fatto che non necessita di normalizzazioni, è la *similarità coseno*, che esprime il valore del coseno dell'angolo $\theta$ che c'è tra i due vettori. Viene calcolato seguendo la formula 2.1: per definizione di coseno, il valore risultante sarà sempre compreso tra $-1$, che definisce i due vettori come aventi la stessa direzione ma verso opposto, e $+1$, che definisce vettori uguali. Tale intervallo di valori permette di confrontare modelli che utilizzano dimensionalità diverse senza normalizzazione.

$$sim_{\cos}(\vec{x}, \vec{y}) = \cos(\theta) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| \cdot |\vec{y}|} \tag{2.1}$$

Si è dunque partiti dallo schema *tf-idf* (*term frequency-inverse document frequency*), introdotto da Salton e McGill [17]. Tale schema prevede un dizionario che contiene tutte le parole (dette anche termini) presenti nel corpus. Per ogni documento del corpus viene effettuato un conteggio del numero di occorrenze di ogni parola; tale conteggio viene chiamato *term frequency count*. Si effettua poi un'opportuna normalizzazione, per poterlo confrontare con un altro conteggio, quello della frequenza delle parole all'interno di tutto il corpus, chiamato *inverse document frequency count*, generalmente in scala logaritmica e a sua volta normalizzato.

La funzione *tf-idf* è scomposta in due fattori: il primo (2.2), *tf*, rappresenta il numero $n_{i,j}$, cioè il numero di occorrenze del termine $i$ nel documento $j$, diviso per $|d_j|$, che è la dimensione del documento $j$, per evitare di dare un peso maggiore ai documenti più lunghi.

$$tf_{i,j} = \frac{n_{i,j}}{|d_j|} \tag{2.2}$$

Il secondo fattore (2.3) indica l'importanza del termine $i$ all'interno del corpus:

$$idf_i = \log \frac{|D|}{|\{d : i \in d\}|} \tag{2.3}$$

Dove $|D|$ è il numero di documenti nella collezione, mentre il denominatore è il numero di documenti che contengono la parola $i$.

L'equazione *tf-idf* può dunque essere scritta come:

$$(tf - idf)_{i,j} = tf_{i,j} * idf_i$$

Il risultato finale è una matrice $X$ parola $\times$ documento, le cui colonne contengono i valori *tf-idf* per ogni documento del corpus. La funzione *tf-idf* riduce dunque documenti di lunghezza arbitraria a vettori numerici a dimensione fissa. Sebbene *tf-idf* abbia delle caratteristiche apprezzabili, legate soprattutto alla capacità di individuare insiemi di parole che identificano un documento, tale approccio non garantisce riduzioni

dimensionali significative e rivela poco sulle relazioni statistiche e strutturali inter- ed intra-documento.

Per risolvere queste mancanze i ricercatori hanno proposto altre tecniche di riduzione della dimensionalità, tra le quali spicca *Latent Semantic Indexing* (LSI), presentato nell'articolo di Deerwester et al. [18]. Questa tecnica prevede l'utilizzo di una decomposizione a valori singolari (*single value decomposition* o SVD) della matrice $X$ per identificare un sottospazio lineare delle caratteristiche *tf-idf* che catturano la maggior parte della varianza nella collezione. Questo approccio permette di raggiungere compressioni decisamente migliori; inoltre l'articolo di Deerwester et al. [18] afferma che LSI permette di individuare alcuni aspetti linguistici di base, come la sinonimia e la polisemia.

Un netto miglioramento da questi punti di vista si è avuto con l'introduzione del modello *probabilistic Latent Semantic Indexing* (pLSI), proposto da Hofmann nel 1999 [19] come alternativa ad LSI. L'approccio pLSI modellizza ogni parola in un documento come un campione di un modello di mistura, dove i componenti della mistura sono variabili polinomiali casuali che possono essere viste come la rappresentazione dei topic. Dunque ogni parola viene generata da un singolo topic, e parole differenti nel documento possono essere generate da topic differenti. Ogni documento viene rappresentato come una lista di proporzioni diverse di componenti della mistura e ridotto così ad una distribuzione di probabilità su un numero fissato di topic. Nonostante pLSI sia un passo avanti nella modellazione probabilistica del testo, si dimostra incompleta non fornendo un modello probabilistico al livello dei documenti: ogni documento è infatti rappresentato come una lista di numeri (le proporzioni della mistura dei topic) e non compare un modello probabilistico generativo per questi numeri. Questo porta a due problemi principali:

- il numero di parametri del modello cresce linearmente con la dimensione del corpus, il che porta a problemi di *overfitting*, che avviene quando un modello probabilistico complesso si adatta ai dati osservati perché si ha un numero di parametri eccessivo rispetto alle osservazioni;
- non è chiaro come assegnare una probabilità ad un documento al di fuori del training set.

## 2.2 LDA

Per migliorare pLSI è stato necessario considerare il concetto alla base delle riduzioni dimensionali proposte da LSI e pLSI, cioè la considerazione del documento come *bag-of-words*: dunque l'ordine delle parole al suo interno è trascurabile. Questo porta al concetto di *scambiabilità*: le parole di un documento sono scambiabili, dunque è possibile mescolarle a piacere.

Il teorema di de Finetti [20] stabilisce che una qualunque collezione di variabili casuali scambiabili viene rappresentata come una mistura di distribuzioni, dunque se si vuole avere una rappresentazione scambiabile di parole e documenti è necessario considerare modelli di mistura che catturino la scambiabilità di entrambi.

Alla base di questa metodologia di pensiero si trovano le radici del modello *Latent Dirichlet Allocation* (LDA), proposto da Blei et al. nel 2003 [21]. L'obiettivo è quello di trovare un modello probabilistico del corpus che non solo assegni un'alta probabilità ai membri del corpus, ma anche ai documenti simili.

LDA è un modello probabilistico generativo di un corpus. L'idea alla base è che i documenti vengano rappresentati come misture casuali sui topic, dove ogni topic è caratterizzato da una distribuzione di parole. LDA assume che ogni documento venga prodotto nella seguente modalità:

- viene deciso il numero di parole $N$ che il documento deve avere, in accordo alla distribuzione di Poisson;
- viene scelta una mistura di topic dall'insieme $\theta$, in base alla distribuzione di Dirichlet su un numero fissato $k$ di topic;
- genera ogni parola $w_i$ nel documento nel seguente modo:
  - sceglie un topic seguendo la distribuzione multinomiale descritta prima;
  - utilizza il topic per generare la parola stessa, in accordo con la distribuzione multinomiale del topic.

Assumendo questo modello generativo per una collezione di documenti, LDA cerca di tornare indietro dai documenti per trovare un insieme di topic probabili che hanno generato il corpus.

Si assuma di avere un insieme di documenti e $k$ topic da definire. LDA per apprendere la rappresentazione di ogni documento in base ai topic e per apprendere le parole associate ad ogni topic effettua i seguenti passi:

- per ogni documento $d$ assegna ogni parola $w_i$ ad uno dei $k$ topic; notare come questo passaggio fornisca sia una rappresentazione del documento basata sui topic, sia una distribuzione delle parole su tutti i topic;
- per migliorare, per ogni documento $d$, calcola per ogni topic $t \in \theta$:
  - $p_t = p(t|d)$, cioè la proporzione di parole nel documento $d$ assegnate al topic $t$;

  - $p_w = p(w|t)$, cioè la proporzione di assegnamenti al topic $t$ su tutti i documenti che arrivano dalla parola $w$;
  - alla parola $w$ viene assegnato un nuovo topic, che abbia probabilità pari a $p_t * p_w$; cioè si sta assumendo che tutti gli assegnamenti di topic siano corretti eccetto quello per la parola in questione, aggiornando solo questo assegnamento.

- il passo precedente viene ripetuto un numero di volte tale da permettere un buon risultato.

Gli assegnamenti mostrati vengono dunque utilizzati per stimare le misture di topic all'interno di un documento e le parole associate ad ogni topic.

### Esempio LDA

In seguito si descrive un piccolo esempio per mostrare il funzionamento (in maniera astratta) di LDA.

Come input, si fornisce al sistema un corpus formato da cinque frasi:

1. Mi piace giocare a basket e calcio.
2. Ieri ho giocato a calcetto al campetto.
3. I tablet e gli smartphone sono costosi.
4. Mia sorella ha comprato un tablet ieri.
5. Oggi guarderò il basket sul tablet.

LDA scopre i topic che queste frasi contengono, ma il numero deve essere fissato dall'utente del sistema. Si è deciso di chiedere al sistema due topic, e LDA produce il seguente output:

- **Frasi 1 e 2**: 100% topic A;
- **Frasi 3 e 4**: 100% topic B;
- **Frase 5**: 60% topic A, 40% topic B;
- **Topic A**: calcio, basket, calcetto, campetto, ...
- **Topic B**: tablet, smartphone, costosi, ...

## 2.3 Metodi *prediction-based*

I metodi visti in precedenza vengono definiti da Baroni et al. [22] come metodi *count-based*, perché lavorano in base al conteggio delle parole all'interno dei documenti o del corpus. I metodi invece descritti in questa parte sono chiamati *prediction-based*, perché si pongono come obiettivo la previsione della parte mancante della frase, come quelli presentati da Mikolov et al. nel 2013 [23] [24].

Come già sottolineato in precedenza, LSI, pLSI e LDA hanno alla base il concetto di *bag-of-words* (BOW), memorizzando le parole come semplici indici all'interno di un dizionario e considerando trascurabile l'ordine delle parole in un documento. Questo approccio fornisce diversi vantaggi, legati soprattutto a semplicità e robustezza; ma tali tecniche hanno raggiunto un massimo a livello di prestazioni che non può più essere migliorato, dovuto anche all'aumento dell'utilizzo di tecnologie che risolvono il problema dell'*automatic speech recognition*. Per automatic speech recognition si intende un insieme di tecnologie atte al riconoscimento automatico della lingua parlata. Spesso infatti le tecniche descritte in precedenza vengono addestrate e testate su corpus testuali di notevoli dimensioni e di ottima qualità (quest'ultima è una caratteristica rara nell'automatic speech recognition).

Nel 2013 Mikolov [23] [24] ha dunque proposto diverse tecniche di rappresentazione vettoriale delle parole, utilizzando corpus notevoli (miliardi di parole nel corpus, milioni di parole nel dizionario). Il task utilizzato per studiare queste tecniche è il *prediction task*, che, data una sequenza di parole, cerca di determinare la parola successiva. Tali modelli hanno mostrato una notevole capacità nell'identificazione della similarità di una parola, permettendo di svolgere il task tramite operazioni algebriche sui vettori che rappresentano le parole.

Tale metodologia non solo permette di comprendere la semplice similarità, ma riesce ad articolare la similarità in diversi livelli. La similarità dunque supera le regolarità linguistiche: Mikolov cita un esempio per mostrare questo fenomeno [23].

Dati i vettori delle parole inglesi "king" (re), "man" (uomo) e "woman" (donna) si è svolta l'operazione algebrica 2.4 il risultato ottenuto è stato un vettore che si è rivelato essere il più simile alla rappresentazione vettoriale della parola inglese "queen" (regina).

$$vector(king) - vector(man) + vector(woman) \tag{2.4}$$

In uno dei suoi articoli [23] Mikolov propone due modelli, chiamati *Word2vec*, per la risoluzione del prediction task:

- *Continuous Bag-of-Words* (CBOW): l'ordine delle parole nello storico non viene considerato durante la proiezione. Il suo funzionamento viene mostrato nella figura 2.1;

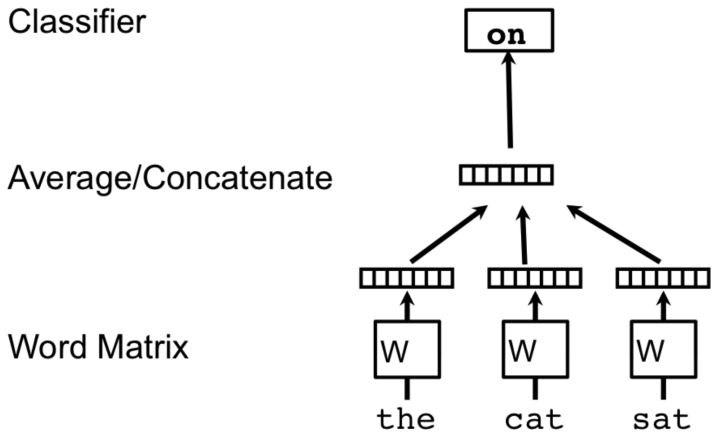


Figura 2.1: Funzionamento del modello CBOW per prevedere la parola `on`.

- *skip-gram*: massimizza la classificazione di una parola in base ad un'altra parola della stessa frase.

In particolare, il modello CBOW implementa una rete neurale (argomento che non viene trattato in questo lavoro di tesi) per creare i vettori delle parole. Nella fase di addestramento viene calcolato l'errore tra l'output del modello ed il valore che l'output avrebbe dovuto assumere: tale errore viene utilizzato per addestrare il modello, che lo riprende come input per regolare alcuni parametri.

Il modello skip-gram durante l'addestramento utilizza invece come input la parola corrente, ed effettua previsioni sulle parole ad essa precedenti e ad essa successive: quelle più lontane avranno una rilevanza minore rispetto a quelle vicine. Il numero di parole da prevedere sia prima che dopo la parola in input viene decisa dal parametro chiamato *context window*, cioè la finestra del contesto della parola.

Skip-gram presenta migliori performance rispetto a CBOW, grazie alle due tecniche che implementa:

- *negative sampling* (campionamento negativo), che prevede di aggiornare solamente una parte dei vettori in output;
- *subsampling*, che prevede di eliminare dall'insieme di addestramento le parole con frequenza troppo alta, che portano meno informazioni rispetto alle parole più rare.

Per essere più formali, nel subsamplig la probabilità di eliminare la parola $w_i$ è calcolata dalla formula 2.5 [24], dove $t$ è una soglia fissata solitamente a $10^{-5}$, mentre $f(w_i)$ è la

frequenza della parola $w_i$.

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{2.5}$$

Dalla rappresentazione vettoriale delle parole, Mikolov ha fatto un ulteriore passo. Come detto in precedenza, i problemi dell'approccio BOW sono diversi, tra i quali spicca però la perdita dell'ordine delle parole. Questo problema porta potenzialmente a rappresentare con gli stessi vettori frasi diverse ma contenenti le stesse parole. Per esporre il concetto in maniera differente, è necessario introdurre il concetto di *contesto*.

Il contesto di una parola è, secondo l'approccio empirista all'elaborazione del linguaggio naturale, l'insieme delle parole che la accompagnano ("*you shall know a word by the company it keeps*" [25]).

Per risolvere il problema di prediction della frase $\bar{w}$ di lunghezza $n$ basterebbe idealmente risolvere l'equazione 2.6.

$$p(\bar{w}) = p(w_1)p(w_2|w_1)\prod_{i=3}^{n} p(w_i|w_1, \ldots, w_{i-1}) \tag{2.6}$$

La produttoria ideale permetterebbe di prendere come contesto tutte le parole della frase precedenti a quella da individuare: naturalmente non è possibile, dunque il contesto viene ridotto.

Uno dei primi modelli presentati per tenere traccia del contesto di una parola era il modello a *n-grammi*, che tiene traccia solamente delle $n-1$ parole precedenti a quella da predire. La produttoria 2.7 mostra un tri-gramma, che dunque guarda come contesto le due parole precedenti a quella da predire.

$$\prod p(w_i|w_{i-2}, w_{i-1}) \tag{2.7}$$

Il modello a n-grammi in sostanza cerca di prevedere la parola successiva attraverso la probabilità condizionata al contesto della parola stessa.

Dal modello a n-grammi è nato l'approccio *bag-of-n-grams* (BOnG) che, come si può intuire, al posto delle parole mappa la parola considerando un breve contesto. Sia BOW che BOnG si sono dimostrati poco sensibili nel catturare la semantica delle parole, o, più formalmente, alla distanza tra i vettori che le rappresentano. Mikolov infatti mostra come le parole "powerful" (potente), "strong" (forte) e "Paris" (Parigi) siano modellizzate da BOW e BOnG come equidistanti, quando è evidente che le prime due parole sono simili tra loro. Secondo Mikolov il contesto permette di catturare meglio queste nozioni di similarità.

I modelli di apprendimento della rappresentazione vettoriale delle parole, come quelli presentati da Mikolov in [23], catturano come mostrato questa similarità. Più formalmente data una sequenza di parole $w_1, w_2, \ldots, w_T$ cercano di massimizzare 2.8.

$$\frac{1}{T}\sum_{t=k}^{T-k} \log p(w_t|w_{t-k}, \ldots, w_{t+k}) \tag{2.8}$$

Alla fine della fase di addestramento le parole con significato simile sono mappate in una posizione simile nello spazio vettoriale. Tali modelli vengono utilizzati per svolgere diversi task nell'ambito dell'elaborazione del linguaggio naturale.

### 2.3.1 Paragraph Vector

*Paragraph Vector* è un algoritmo non supervisionato presentato da Le e Mikolov [26], in grado di dedurre vettori a dimensione fissa da porzioni di testo a dimensione variabile. È ispirato dai modelli di apprendimento della rappresentazione vettoriale delle parole mostrati in questa sezione. I vettori delle parole vengono usati per contribuire alla predizione della parola successiva nella frase. Il punto di partenza è dunque che per predire la parola successiva, i vettori delle parole contengono indirettamente delle informazioni semantiche.

Ogni paragrafo (porzione di testo di dimensione variabile) viene mappato in un unico vettore, rappresentato da una colonna nella matrice $D$, mentre ogni parola viene mappata come una colonna nella matrice $W$. Nel task di previsione il vettore dei paragrafi e i vettori delle parole vengono concatenati oppure viene fatta la media (*concatenate* ed *average*).

La rappresentazione vettoriale del paragrafo viene utilizzata dal sistema esattamente come viene utilizzata la rappresentazione vettoriale delle parole: si comporta come un'informazione aggiuntiva, utilizzata dal sistema durante il task di predizione, che contiene il contesto legato alla parola stessa. I vettori dei paragrafi possono poi essere utilizzati da altre tecniche di machine learning, oppure da algoritmi di clustering (che verranno descritti successivamente). Riassumendo, l'algoritmo ha due passi fondamentali:

- addestramento per creare i vettori delle parole e dei paragrafi/documenti;
- passo di inference, per ricavare i vettori dei paragrafi non ancora visti.

Questo modello viene chiamato *Distributed Memory Model of Paragraph Vectors* (PV-DM), ed il suo funzionamento è mostrato nella figura 2.2.

Esiste una variante a PV-DM, chiamata *Distributed Bag-of-Words* (PV-DBOW), che non utilizza i vettori delle parole ma forza il modello a predire la parola in base a parole

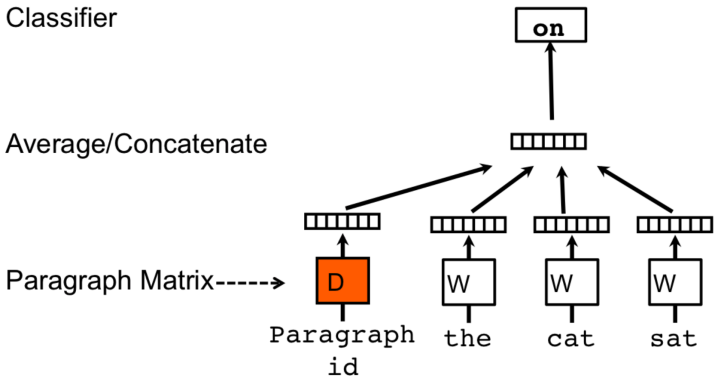


Figura 2.2: Rappresentazione grafica del funzionamento di PV-DM.

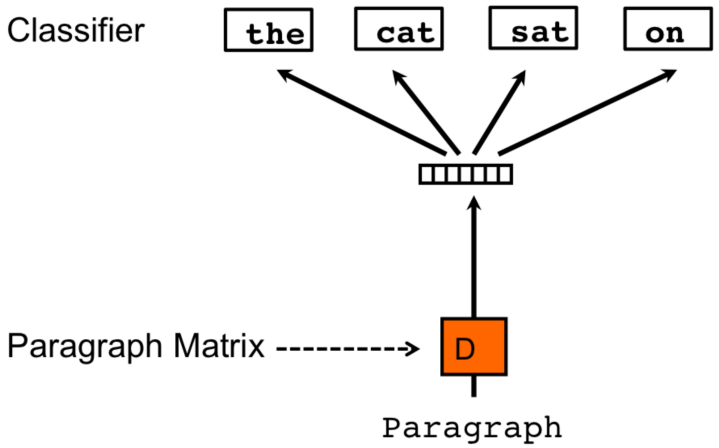


Figura 2.3: Rappresentazione grafica del funzionamento di PV-DBOW.

campionate casualmente all'interno del paragrafo. Il vantaggio di questo modello è che richiede un minor numero di dati da salvare, poiché non deve salvare la matrice dei vettori delle parole. Il suo funzionamento è mostrato nella figura 2.3

I vantaggi di questo algoritmo sono:

- lavora su dati non etichettati, dunque è perfetto per task non supervisionati come topic detection;
- vengono ereditate dai vettori delle parole alcune proprietà, come per esempio la vicinanza semantica.

Per questi pregi è stato scelto come punto centrale dell'implementazione del lavoro di tesi. Nel capitolo riguardante l'implementazione della soluzione si esaminerà nel dettaglio

il suo inserimento all'interno della soluzione, mostrando le librerie software che permettono di utilizzarlo. Poiché il task di topic detection richiede un clustering dei dati, la trasformazione vettoriale dei tweet attraverso Paragraph Vector permette di manipolarli come oggetti geometrici. Nel capitolo seguente si mostreranno le tecniche di clustering utilizzata nella soluzione.

# Capitolo 3

# Clustering

In questo capitolo si descrivono le principali tecniche di clustering utilizzate per l'implementazione del lavoro di tesi. Si ricorda infatti che l'obiettivo è quello di identificare dei cluster di tweet assegnando loro insiemi di storie, creando un sistema che conservi le proprie prestazioni (in termini statistici) anche al di fuori dei dati usati per costruirlo.

Un'ottima introduzione al clustering si trova nel libro scritto da Manning e Schütze, *Foundations of Statistcal Natural Language Processing* [27]. Essa definisce il task di *clustering* come la suddivisione di dati in gruppi omogenei, senza alcuna informazione pregressa.

I dati per essere sottoposti al clustering devono essere rappresentati in maniera coerente: in questo lavoro di tesi viene dunque utilizzato l'output dell'algoritmo Paragraph Vector, visto nella sezione precedente, dove i dati sono rappresentati da un insieme di vettori numerici di lunghezza fissa $N$, distribuiti dunque in uno spazio $N$-dimensionale. I vettori di tweet semanticamente simili sono collocati in posizioni geometricamente vicine.

Il passo successivo alla trasformazione geometrica dei tweet è appunto quello di individuare un opportuno algoritmo di clustering, che permetta dunque di assegnare ad ogni tweet un topic. Da questo momento in poi la nozione di topic sarà equivalente alla nozione di cluster.

Per risolvere il problema di clustering durante lo svolgimento del lavoro di tesi si sono intraprese due strade, estremamente differenti dal punto di vista dell'utilizzo dei vettori dei tweet. Nel primo caso si sono infatti considerati i vettori (dunque i tweet) come nodi all'interno di un grafo pesato. Nel secondo (che sarà quello implementato effettivamente nella soluzione), i tweet saranno visti da punto di vista esclusivamente geometrico.

Prima di procedere alla descrizione degli algoritmi utilizzati per la ricerca di una soluzione al problema presentato da questo lavoro di tesi, è necessario fornire una breve introduzione al clustering, attraverso la definizione delle tipologie di algoritmi di

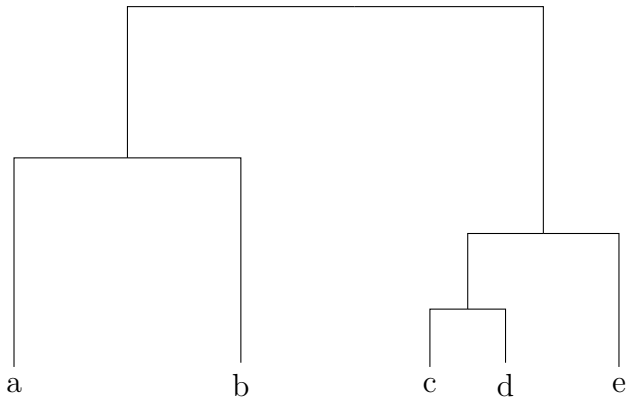


Figura 3.1: Esempio di dendrogramma.

clustering e della funzione di similarità.

## 3.1 Tipologie di algoritmi di clustering

Secondo Manning e Schütze esistono numerosi algoritmi di clustering, che possono però essere ricondotti a poche tipologie di base [27]. La prima categorizzazione avviene in base alle strutture prodotte dagli algoritmi di clustering, la seconda in base alla tipologia di assegnamento del dato al cluster.

Se l'algoritmo produce una partizione dei dati in un certo numero di cluster, l'algoritmo viene detto *flat* o *non gerarchico*; Rousseeuw e Kaufman nel loro libro del 1990 [28] utilizzano anche la denominazione di algoritmi *partitivi*. La maggior parte di questi algoritmi è *iterativa*, cioè parte da un dato insieme di cluster iniziali, che vengono raffinati iterando un'operazione di ricollocamento dei dati all'interno dei cluster.

Se l'algoritmo produce invece una gerarchia, intesa come struttura in cui ogni nodo è una sottoclasse del nodo padre, l'algoritmo di clustering viene detto *gerarchico*. Spesso la gerarchia viene rappresentata da un *dendrogramma*, come mostrato dalla figura 3.1. Un dendrogramma è un diagramma ramificato, dove la similarità tra i nodi è mostrata dall'altezza in cui i loro rami si uniscono: maggiore l'altezza, più la similarità è bassa. Osservando la figura 3.1 si nota allora come $c$ e $d$ abbiano una similarità maggiore rispetto ai cluster $a$ e $b$ Ogni nodo nel dendrogramma rappresenta dunque un cluster, creato dall'unione dei due cluster sottostanti.

La seconda caratteristica per discriminare le tipologie di algoritmi di clustering viene definita in base alla tipologia di assegnamento di un oggetto al cluster. Un algoritmo di clustering di tipo *hard* assegna un vettore ad uno ed un solo cluster. In un algoritmo di clustering di tipo *soft* possono esistere diversi gradi di appartenenza: in alcuni casi

è anche possibile assegnare un oggetto a più cluster. In un modello probabilistico, per esempio, un algoritmo di clustering soft definisce i livelli di appartenenza attraverso $p(c_j|x_i)$, cioè la probabilità che l'oggetto $x_i$ sia un membro del cluster $c_j$. In un modello basato su vettori come quello presentato da questo lavoro di tesi, i gradi di appartenenza possono essere formalizzati attraverso la similarità del vettore con il centro del cluster. Il centro del cluster viene chiamato *centroide* ($\vec{\mu}$), ed è definito come segue:

$$\vec{\mu} = \frac{1}{M} \sum_{\vec{x} \in c} \vec{x}$$

con $M$ cardinalità del cluster.

È necessario aggiungere che spesso gli algoritmi che adottano una tipologia di assegnamento soft assumono che ogni oggetto appartenga ad uno ed un solo cluster. La differenza rispetto agli algoritmi di clustering che adottano una tipologia di assegnamento hard è nell'incertezza sulla correttezza dell'associazione tra oggetto e cluster. Esiste comunque una tipologia di modelli, chiamati *disgiuntivi*, in cui un oggetto può effettivamente essere assegnato a più cluster.

Negli algoritmi di clustering gerarchici si adotta solitamente la tipologia di assegnamento hard; negli algoritmi di clustering partitivi invece si possono trovare entrambe le tipologie.

## 3.2 Funzione di similarità

Il nucleo di ogni algoritmo di clustering si trova nella *funzione di similarità*. Formalmente definisce il concetto di similarità che intercorre tra cluster o oggetti, permettendo di procedere nella generazione dei cluster. Essa viene infatti utilizzata sia durante l'assegnamento di un vettore ad un cluster, sia nel momento di suddividere un cluster in una o più parti sia durante l'unione di due o più cluster.

Spesso si riscontrano delle difficoltà durante gli studi delle funzioni di similarità. Sempre seguendo il testo di Manning e Schütze [27], si apprende come l'ambiguità compaia all'interno della terminologia utilizzata nella definizione della similarità. si vogliono unire oggetti o cluster che abbiano la **massima** similarità. Spesso si trovano definizioni che invece affermano che si vogliono unire oggetti o cluster a distanza **minima**.

Per questo motivo la funzione di similarità tra vettori viene definita in relazione alla nozione di distanza, come mostrato nella formula 3.1. Dati $x$, $y$ vettori, $sim(x, y)$ funzione di similarità tra i due vettori e $d(x, y)$ misura di distanza tra due vettori, la funzione di similarità viene definita come:

$$sim(x, y) = \frac{1}{d(x, y)} \tag{3.1}$$

Non necessariamente $d$ indica la distanza vettoriale: l'unica cosa importante è che sia una *metrica*. Per essere una metrica $d$ deve essere tale che, per ogni vettore $a$, $b$, $c$:

- $d(a, b) \geq 0$
- $d(a, b) = d(b, a)$
- $d(a, b) + d(b, c) \geq d(a, c)$

Per quanto riguarda la funzione di similarità tra clustering, che viene ricavata dalla similarità tra vettori, ne esistono tre tipologie principali:

- similarità *single link*;
- similarità *complete link*;
- similarità *group average*.

La scelta della funzione di similarità appropriata dipende dalla tipologia del problema.

Si parla di similarità *single link* quando la similarità tra due cluster è data dalla similarità dei due elementi (uno per cluster) a distanza minima. Si controllano dunque tutte le coppie di oggetti all'interno dei due cluster, cercando quella con similarità massima. Più formalmente, anche per evitare le ambiguità descritte in precedenza, la similarità *single link* è definita come:

$$sim(c_1, c_2) = max(sim(x, y))$$

con $c_1$ e $c_2$ cluster e $x \in c_1$, $y \in c_2$ vettori. Questo tipo di similarità permette di ottenere un'alta coerenza locale, dove gli oggetti vicini sono tutti nello stesso cluster.

Si parla di similarità *complete link* quando la similarità tra due cluster è data dalla similarità dei due elementi (uno per cluster) a distanza massima. Si controllano dunque tutte le coppie di oggetti all'interno dei due cluster, cercando quelli con similarità minima. Più formalmente, la similarità *complete link* è definita come:

$$sim(c_1, c_2) = min(sim(x, y))$$

con $c_1$ e $c_2$ cluster e $x \in c_1$, $y \in c_2$ vettori. Questo tipo di similarità tende a privilegiare una coerenza globale dei cluster, contrapponendosi alla visione della similarità single link.

Si parla di similarità *group average* quando la similarità tra cluster è ricavata da una media delle similarità tra i vettori interni ai cluster. È a tutti gli effetti un compromesso

tra single e complete link. Preso un cluster $c_j$, la similarità media $S$ tra i suoi vettori è definita come:

$$S(c_j) = \frac{1}{|c_j|(|c_j|-1)} \sum_{x \in c_j} \sum_{x \neq y \in c_j} sim(x, y)$$

Presi due cluster $c_u, c_v \in C$, con $C$ insieme dei cluster, ad ogni iterazione si prendono i due cluster che massimizzano $S(c_u, c_v)$.

## 3.3 Clustering su grafi

Per meglio definire gli algoritmi di clustering sui grafi è necessario introdurre alcune nozioni di teoria dei grafi. Il *survey* presentato da Satu Elisa Schaeffer [29] offre una panoramica completa sia per quanto riguarda le definizioni di base, sia per quanto riguarda l'introduzione a tali algoritmi. Il libro di Cormen, *Introduzione agli algoritmi e strutture dati* offre inoltre in appendice una sezione per le definizioni della teoria dei grafi [30].

Un *grafo* $G$ è una coppia di insiemi $G = (V, E)$, dove $V$ è un insieme di vertici o nodi, mentre $E$ è l'insieme degli archi, i collegamenti tra i nodi.

L'*ordine* $n$ di un grafo $G = (V, E)$ è definito come $n = |V|$, cioè come il numero dei nodi.

Un *grafo orientato* è un grafo in cui le coppie di nodi sono ordinate. Per esempio, in un grafo orientato potrebbe esistere l'arco che va dal nodo $u$ al nodo $v$, ma non necessariamente l'arco che da $v$ va a $u$. Un *grafo non orientato* ha le coppie di nodi degli archi non sono ordinate, quindi se esiste un arco dal nodo $u$ al nodo $v$ è vero anche il viceversa.

Un *grafo pesato* è un grafo che possiede una funzione peso $w : E \to \mathbb{R}$ che assegna un valore numerico ad ogni arco.

Il clustering su grafi individua gruppi di nodi omogenei all'interno di un grafo di qualunque tipo. Nonostante alcuni grafi si presentino, in maniera informale, con dei cluster "evidenti", gli algoritmi di clustering per grafi devono prendere in input un grafo di qualunque tipo.

Prima di procedere è necessario descrivere come è stato creato il grafo dei nodi. Senza entrare nel dettaglio, l'output di Paragraph Vector consiste in un insieme di vettori. Tali vettori, come si vedrà nel capitolo dedicato alla descrizione dell'implementazione della soluzione, sono stati ricavati da un'implementazione software di Paragraph Vector, che permette anche di effettuare diverse operazioni su di esse. Una di queste permette di definire la distanza tra due vettori: con essa si è creato un grafo non orientato e pesato, il cui peso degli archi era dato dalla distanza vettoriale ricavata dalla libreria. L'arco

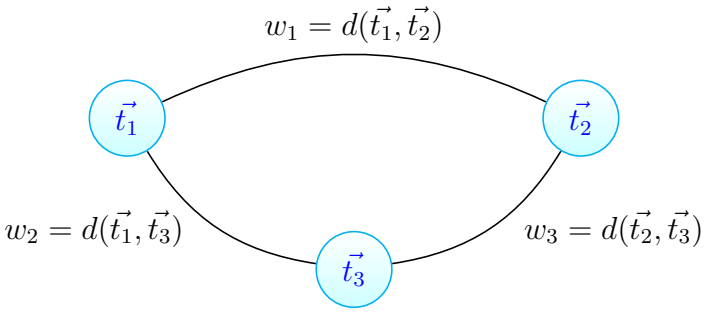


Figura 3.2: Esempio di grafo creato per tre tweet.

tra il nodo $u$ ed il nodo $v$ veniva inserito nel grafo se e solo se $v$ conteneva uno degli $x$ vettori più simili al vettore contenuto in $u$. Il valore di $x$ è stato variato più volte per gestire in maniera più semplice la dimensione del grafo generato.

Nella figura 3.2 si può osservare un esempio minimale di come compaia graficamente un grafo per tre tweet. Ogni nodo contiene il vettore del tweet ricavato da Paragraph Vector, mentre gli archi vengono pesati con la distanza tra i due vettori nel piano $n$-dimensionale.

In seguito verranno descritti i due algoritmi principali utilizzati durante la ricerca del lavoro di tesi, con particolare attenzione ai motivi che hanno portato alla scelta dell'algoritmo oppure al suo abbandono.

### 3.3.1 Affinity Propagation

L'algoritmo di clustering *Affinity Propagation* è stato proposto da Frey e Duek [31] viene trattato all'interno della documentazione della parte di clustering di `scikit-learn` [32], un modulo Python che integra un alto numero di algoritmi di machine learning sia per problemi supervisionati che non [33].

Il punto centrale dell'algoritmo Affinity Propagation è nell'identificazione di un sottoinsieme di "esemplari rappresentativi". In input viene presa una matrice di similarità tra coppie di dati. I dati si scambiano valori reali come messaggi finché non emergono degli esemplari idonei e di conseguenza si ottengono dei buoni cluster.

L'algoritmo è stato scartato durante l'implementazione del lavoro di tesi perché generava troppi cluster e di conseguenza troppi topic anche per insiemi ridotti di tweet.

### 3.3.2 BorderFlow

L'algoritmo *BorderFlow* è stato sviluppato all'interno dell'Università di Lipsia, da Ngomo e Schumacher [34], ed è un algoritmo di clustering soft per grafi di grandi dimensioni, appositamente creato per i problemi legati all'elaborazione del linguaggio naturale.

BorderFlow segue la definizione di cluster proposta da Flake et al. [35], che afferma che un cluster (in un grafo) è un insieme di nodi che hanno più collegamenti tra di loro piuttosto che verso l'esterno. Utilizzando la nozione di flusso, un cluster $X$ può essere visto come un insieme di nodi tali che il flusso all'interno di $X$ sia massimo, mentre il flusso in uscita da $X$ sia minimo. L'idea dietro a BorderFlow e dunque quella di massimizzare il flusso che dai nodi di confine si dirige verso i nodi interni al cluster, minimizzando il flusso che dai nodi interni esce dal cluster. L'algoritmo, descritto nel dettaglio nell'articolo di Ngomo e Schumacher, si basa sull'etichettare i nodi del grafo rispetto al cluster come:

- interni al cluster;
- di confine al cluster;
- appartenenti al vicinato, dunque al di fuori del cluster e collegati ad un nodo interno.

L'algoritmo per generare il valore del flusso considera anche il peso degli archi, dunque può essere utilizzato su grafi pesati.

Anche in questo caso l'algoritmo è stato scartato durante l'implementazione del lavoro di tesi perché generava troppi cluster e di conseguenza troppi topic anche per insiemi ridotti di tweet.

## 3.4 Clustering di vettori

Dopo avere implementato gli algoritmi legati al clustering di grafi, abbandonati per i motivi elencati in precedenza, il lavoro di tesi si è concentrato sull'implementazione di una soluzione che utilizzasse un algoritmo con due caratteristiche principali, cioè che fosse in grado di:

- gestire opportunamente il numero di cluster;
- lavorare direttamente sui vettori estratti da Paragraph Vector.

Il problema principale evidenziato dagli algoritmi di clustering su grafi è stato quello della difficoltà nel gestire grandi moli dati, individuando un eccessivo numero di cluster: per questo il lavoro di tesi si è spostato sulla ricerca di un algoritmo in grado di reggere meglio la dimensione dei dati di input. Si è poi deciso di procedere nella ricerca di un algoritmo in grado di lavorare direttamente sui vettori di output da Paragraph Vector per snellire ulteriormente l'implementazione, evitando la costruzione e la gestione degli archi pesati.

Una soluzione poteva essere quella di utilizzare uno degli algoritmi che richiedono il numero di cluster $k$ in input: tali algoritmi infatti generano $k$ cluster, qualunque sia la dimensione dei dati da partizionare.

L'algoritmo più citato in questa famiglia è sicuramente *K-means*, introdotto dallo studio di James MacQueen del 1967 [36]. Tale algoritmo si basa sulla scelta di $k$ centroidi di partenza: i dati vengono iterativamente associati ad un centroide, permettendo il ricalcolo di quest'ultimo. Alla fine dell'algoritmo, si hanno i $k$ cluster richiesti.

Tale famiglia di algoritmi risolverebbe il problema dell'elevato numero di cluster, ma non è stato possibile utilizzarlo nel contesto del lavoro di tesi a causa della struttura del sistema implementata in SpagoBI. La parte che infatti si occupa della raccolta dei tweet non ne scarica una quantità fissa: per ogni ricerca SpagoBI ottiene un numero molto variabile di tweet, estremamente dipendente dalla scelta della chiave di ricerca, che va dalle poche decine alle diverse migliaia. Tale meccanismo impedisce una opportuna selezione a priori del numero di cluster (il parametro $k$ di K-means). Si potrebbe pensare di calcolare tale parametro quando si conosce il numero di tweet scaricati, ma non si è proceduto in questa direzione ritenendola macchinosa e poco scalabile.

La scelta è ricaduta sull'algoritmo *Hierarchical Density-Based Spatial Clustering of Applications with Noise* (HDBSCAN), uno sviluppo dell'algoritmo *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN). In seguito vengono entrambi descritti nel dettaglio.

### 3.4.1 DBSCAN

L'algoritmo *Density-Based Spatial Clustering of Applications with Noise* o, abbreviato, DBSCAN, è stato presentato nell'articolo di Ester et al. del 1996 [37]. Gli obiettivi che hanno portato allo sviluppo di questo algoritmo sottolineano proprio l'importanza di un algoritmo di clustering in grado di elaborare grandi moli di dati, partizionandoli in cluster di forma arbitraria. Non tutti i punti dell'insieme di partenza vengono assegnati ad un cluster: questi punti vengono chiamati punti di "rumore", e di solito appartengono a zone dello spazio con una bassa densità di punti.

Per questi motivi DBSCAN utilizza una nozione di cluster *density-based* (basato sulla

densità), in grado appunto di generare cluster di forma arbitraria. Per cluster basato sulla densità si intende il definire il cluster come una zona di spazio in cui la concentrazione di punti (dunque la loro densità) è maggiore rispetto all'esterno. Tra i cluster compaiono quelle che sono chiamate le aree di rumore, cioè zone dello spazio in cui i punti non hanno una densità tale da permettergli di far parte di un cluster. Grazie a tali caratteristiche l'algoritmo può essere applicato ad uno spazio di dimensionalità arbitraria, quindi anche per l'insieme dei vettori generati da Paragraph Vector.

Il funzionamento dell'algoritmo è basato sulla localizzazione del vicinato per ogni punto: dato un punto, un certo raggio ed un numero minimo di punti, si individua un vicinato intorno al punto utilizzando il raggio: se il vicinato contiene almeno il numero minimo di punti, allora lo si può considerare un cluster. Si devono dunque dare le seguenti definizioni.

L'$\varepsilon$-vicinato di un punto $p$, denotato da $N_\varepsilon(p)$ è definito come:

$$N_\varepsilon(p) = \{q \in D \big| d(p,q) \leq \varepsilon\}$$

dove $D$ è l'insieme dei punti e $d(p,q)$ è una metrica di distanza tra i punti $p$ e $q$.

Viene richiesto che per ogni punto $p$ in un cluster $C$ esista un punto $q \in C$ tale che $p$ sia all'interno di $N_\varepsilon(q)$ e che $MinPts = |N_\varepsilon(q)|$. $MinPts$ è il numero minimo di punti che un $\varepsilon$-vicinato deve contenere per poter essere considerato un cluster.

I due parametri da fornire in input all'algoritmo sono dunque:

- $\varepsilon$, che determina il raggio che definisce l'$\varepsilon$-vicinato di un punto;
- $MinPts$, che definisce il numero minimo di punti per cui un $\varepsilon$-vicinato possa essere considerato un cluster.

L'algoritmo DBSCAN inizia prendendo un punto $p$ casuale, ricava $N_\varepsilon(p)$ e controlla che abbia almeno $MinPts$ punti. Se è così, viene creato un nuovo cluster; altrimenti $p$ viene etichettato come rumore. L'algoritmo passa poi ad un nuovo punto scelto casualmente e non ancora visitato. Il punto $p$ etichettato come rumore potrebbe comunque entrare in un altro cluster: per esempio, preso il punto $q$, se $|N_\varepsilon(q)| \geq MinPts \wedge p \in N_\varepsilon(q)$ allora $p$ viene etichettato come appartenente al cluster di $q$. L'algoritmo procede finché tutti i punti non sono stati etichettati.

### 3.4.2 HDBSCAN

L'algoritmo *Hierarchical Density-Based Spatial Clustering of Applications with Noise* o, abbreviato, HDBSCAN, è stato presentato nell'articolo di Li e Xi del 2011 [38] e successivamente analizzato da Sun [39]. HDBSCAN è a tutti gli effetti la versione gerarchica

dell'algoritmo DBSCAN descritto precedentemente. Anche in questo algoritmo dunque alcuni punti possono rimanere fuori dal clustering, e vengono etichettati dunque come rumore.

Esattamente come DBSCAN, HDBSCAN necessita di due parametri:

- $\varepsilon$, che rappresenta il raggio dell'$\varepsilon$-vicinato;
- $MinPts$, che identifica il numero minimo di punti richiesti all'$\varepsilon$-vicinato per essere definito come cluster.

L'algoritmo può essere suddiviso in due fasi principali:

1. generazione dei cluster iniziali;
2. unione gerarchica dei cluster iniziali.

La prima fase parte con la selezione di un punto casuale $p$, per il quale viene individuato l'$\varepsilon$-vicinato; se esso contiene almeno $MinPts$ punti, viene formato un nuovo cluster, etichettando $p$ come *core point* ed assegnando sia a $p$ che a tutti i punti dell'$\varepsilon$-vicinato di $p$ un cluster ID. Se $p$ non è un core point e $N_\varepsilon(p)$ è vuoto, $p$ viene etichettato come rumore; altrimenti l'algoritmo testa, per ogni punto $q \in N_\varepsilon(p)$, i diversi $N_\varepsilon(q)$. Se in essi non trova altri core point, $p$ viene etichettato come rumore; altrimenti $p$ prende il cluster ID del core point e viene etichettato come *boundary point* (punto di confine). In questa fase l'algoritmo determina dunque i cluster iniziali, eliminando definitivamente il rumore; essa viene ripetuta finché tutti i punti non vengono etichettati.

La seconda fase unisce i cluster creati dalla prima fase. Vengono analizzati i punti etichettati con più di un cluster ID: si determina per ognuno di essi se è un core point o un boundary point. Quando due o più cluster condividono almeno un core point gli elementi dei cluster vengono uniti in un unico cluster ed etichettati con un unico cluster ID. Se i punti in comune sono tutti boundary point, l'algoritmo confronta, per ogni punto, le distanze tra il punto ed i core point del proprio $\varepsilon$-vicinato, assegnando il punto al cluster più vicino. Questa fase viene permette all'algoritmo di correggere una scelta errata del valore di $\varepsilon$, e viene ripetuta finché ogni punto non è etichettato da un solo cluster ID.

Questo algoritmo è stato scelto per essere implementato all'interno della soluzione perché:

- elimina il problema dell'alto numero di cluster;
- è controllabile facilmente grazie a due soli parametri dal significato intuitivo;

- permette di lavorare direttamente sui vettori in output a Paragraph Vector.

Nel capitolo successivo verrà descritta nel dettaglio l'implementazione utilizzata.

# Capitolo 4

# Architettura e strumenti software

In questo capitolo verranno presentate nel dettaglio le caratteristiche dell'implementazione della soluzione, descrivendone l'architettura ed i meccanismi utilizzati ed elencando i diversi strumenti software utilizzati durante lo sviluppo.

## 4.1 Architettura della soluzione

Come già accennato nei capitoli precedenti, il lavoro di tesi si concentra sulla modifica della parte di topic detection del modulo di social network analysis del software open source SpagoBI [9]. Tale componente si occupa della connessione con Twitter per ottenere i tweet e del loro immagazzinamento in un database relazione MySQL.

MySQL [40] è un *Relational Database Management System* (RDBMS) sviluppato da Oracle, cioè un software per la creazione, la manipolazione e l'interrogazione di un database di tipo relazionale.

### 4.1.1 Twitter

Leggendo Wikipedia [41],Twitter è una rete sociale che fornisce agli utenti, attraverso l'omonima piattaforma, una pagina personale aggiornabile tramite messaggi di testo con lunghezza massima di 140 caratteri; tale soglia è stata da poco portata a 280 caratteri poiché, come spiegato in un articolo del blog di Twitter [42], molti utenti abbandonavano spesso la scrittura del tweet accorgendosi che 140 caratteri non sarebbero bastati.

Un *tweet* è un aggiornamento scritto da un utente sulla propria pagina personale, leggibile da tutti coloro iscritti alla pagina stessa.

Un *retweet* è la condivisione di un tweet. Talvolta gli utenti scrivono "RT" all'inizio di un Tweet per indicare che stanno pubblicando il contenuto di qualcun altro. In questo caso non utilizzano un comando o una funzione di Twitter, ma significa che stanno citando il tweet di un altro utente [43].

Una *menzione* è un tweet che contiene il nome utente di un altro account Twitter, preceduto dal simbolo @ [44].

Un *hashtag*, rappresentato dal simbolo #, viene utilizzato per indicare parole chiave o argomenti su Twitter. Questa funzione è nata su Twitter e permette agli utenti di seguire gli argomenti a cui sono interessati, poiché vengono utilizzati prima di una parola chiave per etichettare il tweet stesso [45].

I *follower* di un utente $A$ sono gli utenti che ricevono i tweet scritti da $A$ [46].

In ambiti propri dell'information retrieval viene preferito Twitter grazie alla maggiore uniformità dei dati, poiché ogni tweet è vincolato al numero massimo di caratteri, cioè 140.

### 4.1.2 SpagoBI

Una buona introduzione a SpagoBI viene fornita nel capitolo dedicato a questo software nel libro *Sentiment Analysis in Social Networks* [47].

SpagoBI è una suite di business intelligence open source sviluppata dal gruppo Engineering [48], e copre diverse aree della business intelligence con un insieme di strumenti analitici e funzionalità adattate ai diversi domini della business intelligence. SpagoBI viene rilasciata con la licenza Mozilla Public License versione 2.0 [49], perciò chiunque può studiare, utilizzare, modificare e distribuire copie modificate di SpagoBI, purché si attenga alle indicazioni della licenza.

Recentemente è stato sostituito da un nuovo software, Knowage [50], che oltre a mantenere le stesse caratteristiche di SpagoBI a livello di funzionalità e licenze, presenta alcune novità.

SpagoBI fornisce un modulo specifico per l'ascolto dei social network, chiamato modulo di social network analysis. L'utente può effettuare analisi sia sui dati in tempo reale sia su dati storici, specificando l'intervallo di tempo. Lo scopo principale del modulo di social network analysis è di osservare i tweet che corrispondono a determinati account o parole chiave per analizzarli e fornire all'utente sia una serie di informazioni di base, sia analisi più avanzate come il topic detection e il sentiment analysis. Gli script che si offrono di fornire le informazioni di base e i risultati delle analisi più complesse sono scritti nel linguaggio di programmazione R, un linguaggio specificatamente creato per task grafici e calcoli statistici [51].

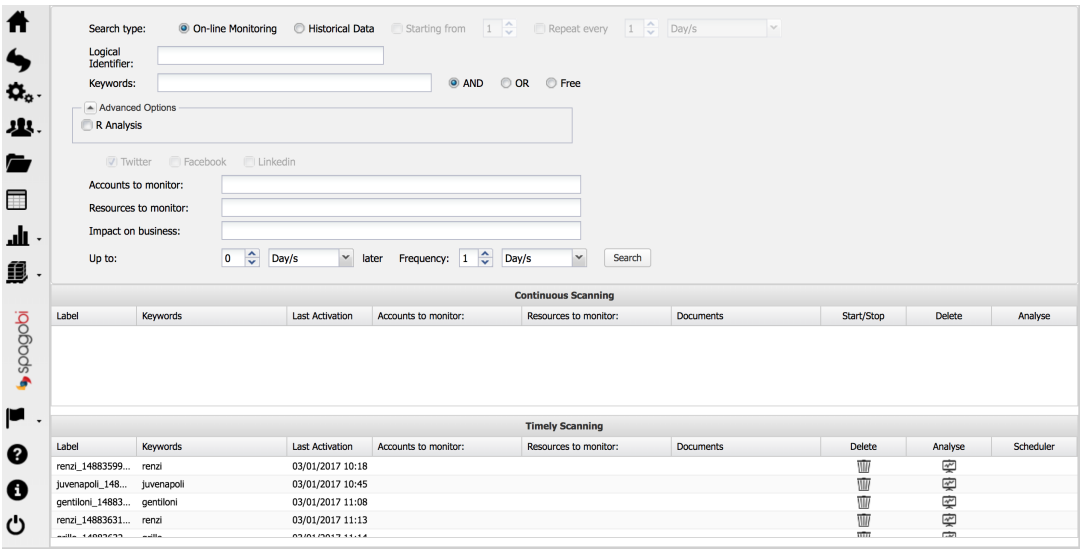


Figura 4.1: Twitter Search Form di SpagoBI

Il modulo permette dunque all'utente di cercare tweet senza la necessità di utilizzare l'interfaccia di programmazione *Twitter Search*, poiché gestisce direttamente la connessione con il social network. L'utente si deve limitare alla registrazione dell'applicazione sul sito *Twitter Application Management* [52] per ottenere le chiavi e i token (*consumerKey*, *consumerSecret*, *accessToken*, *accessTokenSecret*) necessari alla configurazione del modulo.

In questo modo l'utente può avviare la ricerca dei tweet tramite il *Twitter Search Form*, mostrato nella figura 4.1.

In questo form è possibile scegliere tra due modalità di ricerca:

1. *streaming*: i tweet che contengono le caratteristiche indicate dall'utente vengono scaricati in tempo reale, dal momento in cui viene avviata la ricerca fino all'interruzione da parte dell'utente stesso;

2. *historical*: i tweet che contengono le caratteristiche indicate dall'utente vengono scaricati tra i tweet indicizzati in un periodo di tempo che può essere standard (da 6 a 9 giorni prima) o specificato dall'utente.

Nel form l'utente può anche specificare risorse ed account da controllare, per esempio per osservare gli andamenti dei follower nel tempo, contare il numero di click su un determinato link, etc.

L'andamento della ricerca può essere controllato tramite due pannelli, uno per le ricerche in tempo reale ed uno per quelle storiche. Una volta conclusa la ricerca l'utente può visualizzare i risultati nelle tab denominate *Twitter Result*, mostrate nella figura 4.2.

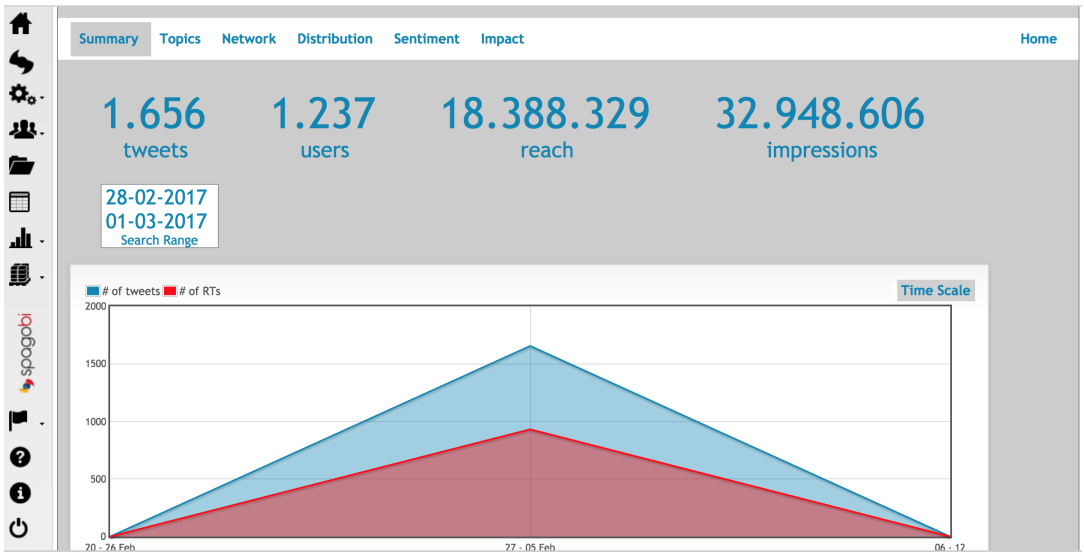


Figura 4.2: Twitter Result di SpagoBI

In particolare l'utente può visualizzare la rappresentazione grafica:

- delle timeline dei tweet;
- delle origini dei tweet;
- dei tweet più rilevanti e più recenti;
- dei topic di cui trattano i tweet;
- delle informazioni su chi ha pubblicato i tweet, con l'aiuto di un grafo che descrive le interazioni tra utenti;
- della distribuzione geografica dei tweet;
- della polarità dei tweet (sentiment analysis).

Il funzionamento del modulo di social network analysis di SpagoBI viene descritto seguendo una semplificazione grafica, mostrata dalla figura 4.3. Seguendo i numeri apposti sulle frecce si segue il processo che porta ad etichettare i tweet in base al topic:

1. viene creata una connessione con Twitter per scaricare i tweet collegati alla chiave di ricerca scelta dall'utente; per fare ciò il programma ha bisogno delle chiavi e dei token dell'utente, che regolano i limiti di tweet scaricabili;
2. vengono scaricati i tweet restituiti da Twitter in base alla chiave di ricerca;

3. i tweet scaricati vengono inseriti nella tabella *TWITTER_DATA* del database MySQL;

4. vengono lanciati gli script R per il topic detection e il sentiment analysis, rispettivamente chiamati *Topic_Modeling_Twitter_TM_outputDB_unificato.r* e *Sentiment_Analysis_Twitter_DB_unificato.r*, utilizzando come identificativo dei tweet appena scaricati il campo *SEARCH_ID* della tabella, che identifica con un numero progressivo le ricerche effettuate dal sistema;

5. concentrandosi sullo script preposto al topic detection, esso scarica i tweet con la corretta *SEARCH_ID* ed effettua un calcolo del numero ottimale di topic; viene poi utilizzato l'algoritmo LDA [21] per assegnare ad ogni tweet il proprio topic;

6. lo script R aggiorna l'apposito campo *TOPICS* della tabella del database, assegnando dunque ad ogni tweet l'etichetta determinata dall'algoritmo LDA;

7. SpagoBI mostra poi all'utente, in maniera grafica ed interattiva, i risultati ricavati nell'apposita dashboard.

Finito questo procedimento l'analisi può considerarsi conclusa. Per approfondire il funzionamento del modulo di social network analysis di SpagoBI si rimanda alla documentazione di SpagoBI [53].

Per meglio comprendere il funzionamento del sistema è opportuno descrivere brevemente i campi interessanti della tabella *TWITTER_DATA* presente all'interno del database MySQL utilizzato da SpagoBI:

- *TWEET_ID*: identificativo numerico univoco utilizzato per definire un tweet;
- *SEARCH_ID*: identificativo numerico progressivo che definisce a quale ricerca effettuata dall'utente è collegato il tweet;
- *TWEET_TEXT*: testo del tweet;
- *TOPICS*: concluso il topic detection, questo campo contiene l'identificativo del topic a cui appartiene;
- *IS_POSITIVE*, *IS_NEUTRAL*, *IS_NEGATIVE*: conclusa la sentiment analysis, questi *flag* vengono utilizzati dal sistema per definire rispettivamente se un tweet è positivo, neutrale o negativo.

La soluzione proposta da questo lavoro di tesi si va a collocare dopo il passo in cui viene chiamato lo script *Topic_Modeling_Twitter_TM_outputDB_unificato.r* Tale script è stato infatti modificato per mantenere intatta la struttura delle varie componenti di

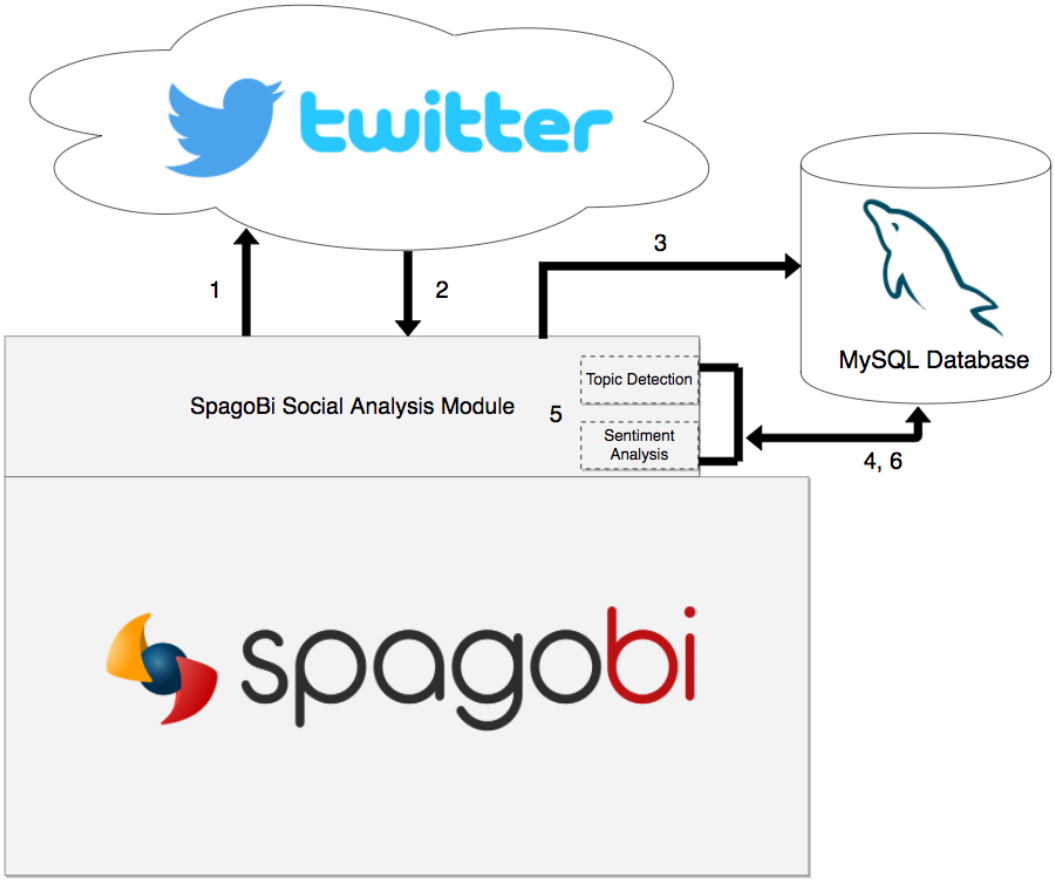


Figura 4.3: Schema architetturale del modulo di social network analysis implementato in SpagoBI

SpagoBI, ma le sue funzionalità sono state tutte eliminate, trasformandolo in un file che si occupa solamente della chiamata allo script Python che implementa l'algoritmo di topic detection proposto da questo lavoro di tesi. Osservando la figura 4.4 si possono notare le differenze con il sistema di SpagoBI. La soluzione dunque affronta i seguenti passaggi:

1. SpagoBI lancia lo script *Topic_Modeling_Twitter_TM_outputDB_unificato.r*;
2. lo script R si limita solamente a chiamare a sua volta lo script Python *topic_detection.py*, specificando il valore *SEARCH_ID*;
3. lo script Python recupera i tweet della ricerca utilizzando per la query al database il valore *SEARCH_ID*;
4. per ogni tweet viene fatta un'operazione di inference utilizzando il modello Paragraph Vector, la cui creazione verrà descritta nel prossimo capitolo;
5. tale operazione permette di ottenere un vettore del tweet coerente con il modello Paragraph Vector;
6. ottenuti tutti i vettori, lo script Python si occupa di utilizzare l'algoritmo HDBSCAN per clusterizzare i vettori, dunque per assegnare ad ogni tweet il proprio topic;
7. la colonna *TOPICS* del database viene aggiornata per permettere di mostrare graficamente all'utente i risultati di HDBSCAN.

Il contenuto del nuovo script R *Topic_Modeling_Twitter_TM_outputDB_unificato.r*, che ha mantenuto lo stesso nome del suo predecessore, viene dunque ridotto alle righe seguenti:

```
library(RMySQL)
library(DBI)
library(tau)
library(tm)
library(plyr)

search_id <-$P{search_id}

closeAllConnections()
unlink(paste("Topic_Modeling","_",search_id,".log",sep=""))
sink.reset()
con_log<-file(paste("Topic_Modeling","_",search_id,".log",sep=""))
sink(con_log,append=TRUE,type="output")
sink(con_log,append=TRUE,type="message")
print(paste("Log of last execution:",Sys.time()))
```

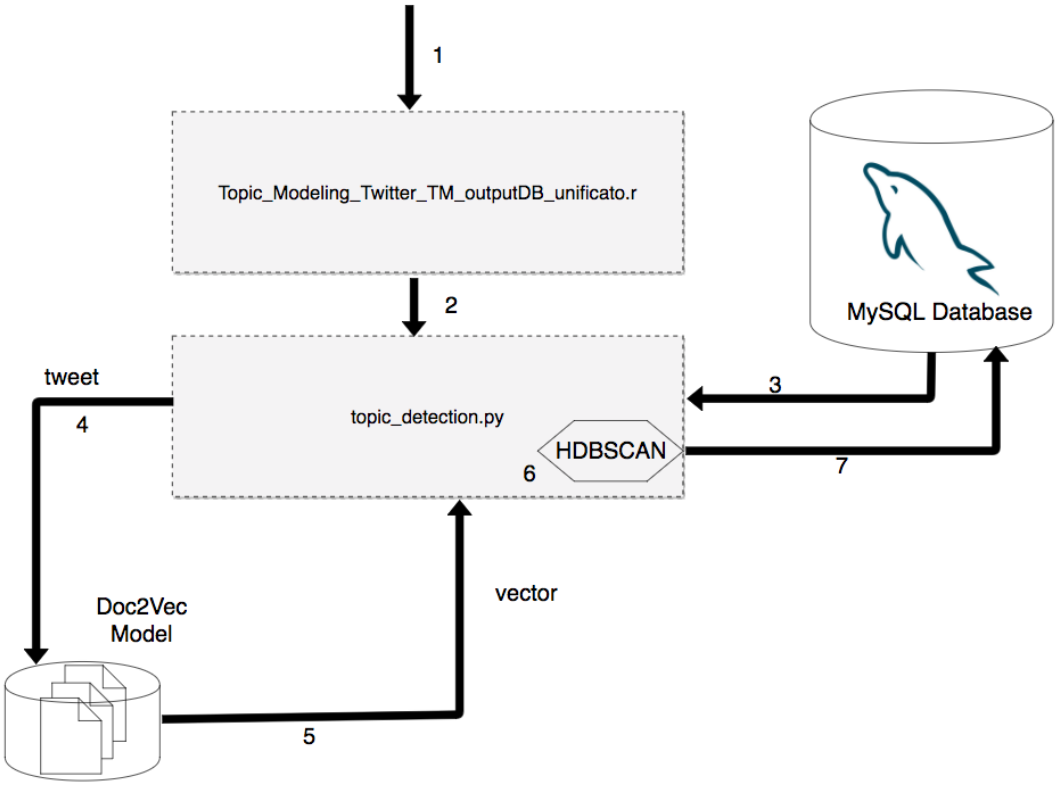


Figura 4.4: Schema architetturale delle modifiche rispetto al sistema di social network analysis implementato in SpagoBI

```
16 print(paste("R home:",R.home()))
17
18 system(paste('python3 topic_detection.py ', search_id, sep=""))
19 print(paste("Topic detection done:", Sys.time()))
```

Si noti dunque come questo script si occupi solamente della creazione di un nuovo file di log e di lanciare lo script Python preposto al topic detection specificando il valore di `search_id`, che permette di recuperare i tweet corrispondenti alla ricerca dell'utente.

## 4.2 Strumenti software

In seguito vengono presentati i diversi strumenti software utilizzati durante l'implementazione della soluzione proposta da questo lavoro di tesi. In particolare si porrà l'attenzione sulla libreria *Gensim* e sulle API utilizzate per implementare l'algoritmo HDBSCAN.

La soluzione, eccetto nella parte R utilizzata solamente per avviare la topic detection, è stata sviluppata in Python 3.6. Le novità rispetto alla versione precedente di Python vengono elencate nel sito della documentazione del linguaggio [54]; la versione effettivamente utilizzata è la 3.6.1 [55], derivata dalle correzioni effettuate alla versione 3.6.0 [56].

Il linguaggio Python è stato scelto per diverse motivazioni, che corrispondono a quelle elencate da Oliphant [57], uno dei creatori di *SciPy*, strumento software utilizzato durante lo sviluppo del lavoro di tesi che verrà descritto successivamente. Nel giustificare la scelta di Python come linguaggio per creare uno dei framework più importanti nell'ambito del calcolo numerico, del machine learning e nell'analisi dei dati, vengono elencate le seguenti cause, sicuramente compatibili con le necessità di questo lavoro di tesi:

- licenza open source che permette di condividere liberamente il proprio codice;
- multipiattaforma;
- sintassi semplice e pulita;
- possibilità di estendere con semplicità il linguaggio;
- possibilità di inglobare codice Python in applicazioni scritte in altri linguaggi, come per esempio SpagoBI, sviluppato in Java;
- l'elevato numero di librerie per effettuare task di elaborazione del linguaggio naturale, manipolazione complessa di vettori e numeri reali; in particolare contenenti sia l'implementazione di Paragraph Vector, sia di HDBSCAN.

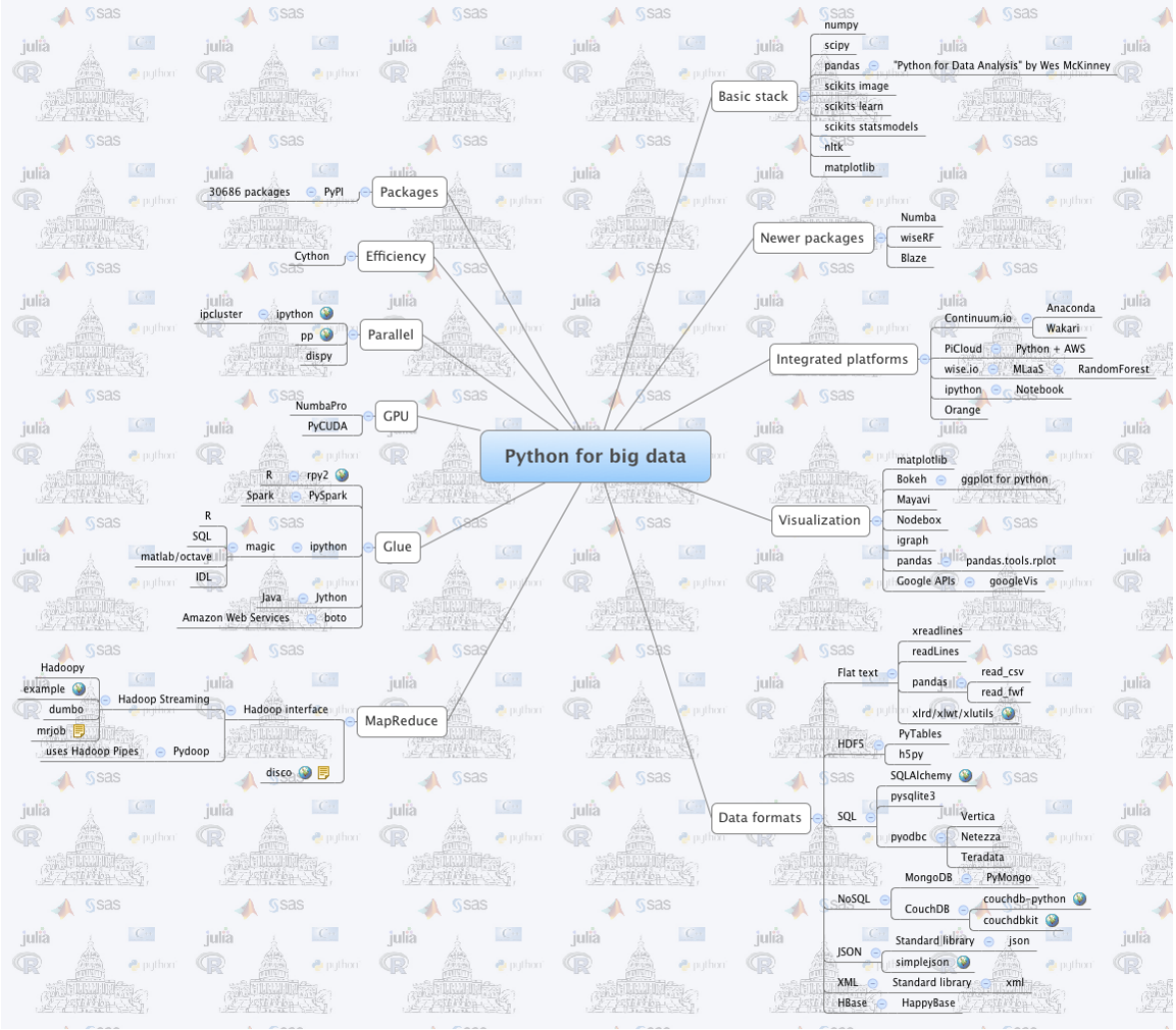


Figura 4.5: Strumenti software collegati all'analisi dei dati per Python.

Come citato infatti nella prefazione del libro di Idris sull'analisi dei dati correlata a Python [58], "*data analysis is Python's killer app*". Sapendo che una *killer app* è un applicativo che da solo determina l'utilizzo di un sistema (hardware o software), si comprende come questa affermazioni consideri obbligatoria la scelta del Python per risolvere problemi legati all'analisi dei dati.

Questo naturalmente non vuol dire che altri linguaggi non posseggano strumenti per risolvere gli stessi task: Python è stato scelto perché permette un'integrazione immediata e semplice delle varie componenti. Nella figura 4.5 viene mostrata una mappa prodotta con il software XMind [59], che mostra appunto l'elevato numero di librerie per l'analisi dei dati in Python.

In seguito si descrivono gli strumenti software utilizzati durante lo sviluppo della soluzione al problema presentato dal lavoro di tesi.

### 4.2.1 *Gensim*

*Gensim* è una libreria Python nata per risolvere due problemi principali [60]:

- organizzazione e ricerca per similarità su collezioni di documenti digitali;
- implementazioni veloci, efficienti a livello di memoria e scalabili di algoritmi per *single value decomposition* e LDA.

Le due tematiche sono correlate dall'analisi semantica non supervisionata di documenti testuali presenti in collezioni digitali.

Gensim è stata sviluppata da Radim Řehůřek e Petr Sojka, del Dipartimento di Informatica della Masaryk University di Brno (Repubblica Ceca), ed è stata presentata con l'articolo del 2010 scritto dagli stessi sviluppatori [61]. Successivamente allo studio di diversi pacchetti software per l'analisi dei dati, gli autori hanno stilato una lista di difetti che li accomunano:

- la maggior parte di questi software è nata per svolgere task supervisionati, mentre la topic detection è un task non supervisionato;
- questi pacchetti software non sono in grado di gestire corpora di dimensione maggiore alla dimensione della RAM dell'utente, poiché sono strutturate in modo tale da caricare tutto in essa: ciò limita notevolmente l'utilizzo dei pacchetti stessi;
- tali pacchetti software sono stati sviluppati partendo da problematiche legate ad altre discipline (fisica, neuroscienze, elaborazione di immagini, etc.), ma né dall'elaborazione del linguaggio naturale, né dall'information retrieval;
- questi pacchetti software coprono un alto numero di algoritmi, generando interfacce complesse e dipendenze difficili da gestire.

Secondo gli autori, l'ultimo punto in particolare ha portato alla nascita di numerosissimi software: la curva di apprendimento per portare il proprio contributo ad uno dei grossi progetti, intricati e complessi da gestire, può aver portato molti programmatori a considerare più semplice uno sviluppo "indipendente" del proprio software, creando dunque una fitta nuvola di pacchetti preposti all'analisi dei dati.

Le considerazioni alla base della nascita di Gensim partono dalla volontà di risolvere questi difetti, e possono essere riassunti nei punti seguenti:

- indipendenza dalla dimensione del corpus, fornendo un framework che permetta di lavorare con dati di dimensione superiore alla RAM, dunque senza caricarli tutti;
- minimizzazione del numero di nomi di metodi ed interfacce che necessitano di essere memorizzati per l'utilizzo del framework;
- utilizzo di una terminologia derivata completamente dall'elaborazione del linguaggio naturale e dall'information retrieval per denominare i metodi e le interfacce;
- necessità di avere un sistema in grado di lavorare senza troppe dipendenze, semplice da installare e che non richieda permessi di amministrazione o installazioni invasive all'interno del sistema ospite;
- copertura degli algoritmi più diffusi, con particolare attenzione all'ambito della topic detection.

Anche gli sviluppatori di Gensim hanno scelto Python come linguaggio per sviluppare la libreria, per la sua sintassi compatta, la semplicità e la possibilità di essere installato su più sistemi; inoltre per il Python esiste la libreria *NumPy*, largamente utilizzata nell'implementazione di numerosi componenti di gensim.

La libreria Gensim è stata utilizzata in tutte le parti che necessitano di creare e manipolare i vettori, in particolare:

- durante l'addestramento del modello Paragraph Vector con il corpus di partenza;
- durante l'operazione di inference dei tweet all'interno del modello.

Il modulo di Gensim che contiene l'implementazione di Paragraph Vector è chiamato *Doc2Vec* e viene descritta all'interno della documentazione della libreria [62]. Radim Řehůřek ha anche prodotto un tutorial per mostrare l'utilizzo di base delle funzionalità del modulo [63].

Sempre seguendo l'articolo scritto da Radim Řehůřek e Petr Sojka [61], vengono esposti i due concetti principali del framework, attraverso due esempi.

Il primo concetto fondamentale è quello del *document streaming*. Un corpus è una sequenza di documenti: ogni documento viene elaborato in successione, dunque non richiede mai una completa memorizzazione in RAM del corpus. Un *corpus* viene dunque definito come un oggetto iterabile, che restituisce un insieme di documenti:

```
1 for document in corpus:
2   pass
```

Le caratteristiche di un *documento* vengono rappresentate da un vettore sparso, anch'esso definito come un oggetto iterabile:

```
for fieldId, fieldValue in document:
  pass
```

Questo primo concetto chiave racchiude le caratteristiche desiderate durante la creazione della libreria, in particolare per quanto riguarda l'utilizzo di una terminologia immediata, la semplicità d'uso e l'indipendenza dalla dimensione della RAM dell'utente. Quest'ultimo fattore permette di costruire scenari più ricchi di informazioni, dunque più vicini al mondo reale. Sia corpora che documenti non sono confinati in questa interfaccia: è possibile arricchirli con metodi o attributi, per esempio identificatori, calcoli statistici, tag o altro.

Il secondo concetto chiave è quello della *trasformazione*. Dove un corpus rappresenta i dati, la trasformazione rappresenta il processo di traduzione dei documenti da uno spazio vettoriale ad un altro. Il codice seguente mostra dunque come creare un modello LDA:

```
from gensim.models import LdaModel

lda = LdaModel(corpus, numTopics = 2)
lda[new_document]
```

I dettagli sull'utilizzo del modulo *Doc2vec* verranno esposti nel capitolo successivo, mano a mano che verranno incontrati nel codice dell'implementazione della soluzione.

### 4.2.2 *SciPy*

Come indicato dal sito dagli stessi sviluppatori [64], con *SciPy* si intendono diverse cose: nel caso di questo lavoro di tesi si parla di SciPy come di un "ecosistema", basato su poche componenti software di base ma integrato da numerosi pacchetti secondari. Le componenti considerate di base da SciPy sono:

- il linguaggio Python;
- *NumPy*, il pacchetto centrale per la computazione numerica, che verrà descritto nel dettaglio successivamente;
- la libreria SciPy, di fatto una collezione di algoritmi numerici e tool per diversi domini (elaborazione dei segnali, statistica, etc.);
- *Matplotlib*, un pacchetto che fornisce funzionalità avanzate per il disegno 2D e funzionalità basilari per il disegno 3D.

Intorno a questa base l'ecosistema SciPy viene costruito integrando diversi tool per la gestione e la computazione dei dati ed altre funzionalità che non vengono affrontate durante la discussione di questo lavoro di tesi. Di questi, è importante menzionare:

- *pandas*, che fornisce strutture dati performanti e semplici da usare;
- *scikit-learn* una collezione di algoritmi e tool per il machine learning [**?**];
- numerose librerie *stand-alone* come *NetworkX*, per la gestione di grafi, e *hdbscan*, contenente l'implementazione dell'algoritmo di clustering HDBSCAN, descritte successivamente nel dettaglio.

### *NumPy*

Come accennato in precedenza, uno dei moduli più importanti per l'ecosistema SciPy è NumPy [65], presentato nel 2006 da Oliphant [57].

NumPy è una libreria che fornisce array multidimensionali, costruibili con tipi di dati arbitrari. Si consideri l'esempio sottostante, che riassume in poche righe di codice le funzionalità principali di NumPy:

```
import numpy as N

dt = N.dtype([('id', 'i4'), ('name', 'S12'), ('scores', 'u1', 4)])

a = N.array([(1001, 'James', [100,98,97,60]),
     (1002, 'Kathy', [100,100,85,98]),
     (1003, 'Michael', [84,75, 98,100]),
     (1004, 'John', [84,76,82,92])], dtype=dt)

print(a['name'])
print(a['scores'])
```

L'output al codice precedente mostra come sia semplice manipolare questi array multidimensionali:

```
array(['James', 'Kathy', 'Michael', 'John'], dtype='|S12')
array([[100, 98, 97, 60],
  [100, 100, 85, 98],
  [84, 75, 98, 100],
  [84, 76, 82, 92]], dtype=uint8)
```

Gli array in NumPy hanno metodi ed attributi, come mostrato dall'esempio seguente:

```
print(a.shape)
print(a.ndim)

a.shape = (2, -1)

print(a.shape)
print(a.ndim)
```

che restituisce il seguente output:

```
(4,)
1
(2, 2)
2
```

Questo breve esempio descrive la sintassi delle operazioni elementari per manipolare gli array NumPy. In particolare, seguendo la documentazione di SciPy e NumPy [66]:

- l'attributo `shape` restituisce una tupla contenente le dimensioni dell'array;
- l'attributo `ndim` restituisce il numero delle dimensioni dell'array;
- il metodo `shape` viene utilizzata per cambiare dimensione all'array, a patto che ciò non richieda un cambiamento nel numero di elementi.

NumPy è stato utilizzato nell'implementazione della soluzione proposta in questo lavoro di tesi per la manipolazione dei vettori dei tweet, poiché incluso nell'implementazione di Gensim.

#### scikit-learn

Come accennato in precedenza, *scikit-learn* è un modulo Python che integra un gran numero di algoritmi di machine learning sia per problemi supervisionati che non [33]. Questo modulo è stato utilizzato in diverse occasioni durante il lavoro di tesi, in particolare per quanto riguarda l'implementazione degli algoritmi di clustering, implementati nel modulo *scikit.clustering* [67].

Ogni algoritmo di clustering viene fornito in due varianti:

- attraverso una classe che implementa il metodo `fit`, cioè quello che permette di prevedere il cluster in base al training set; le etichette sono contenute nell'attributo `labels_`;
- attraverso una funzione che, dato il modello addestrato, restituisce un array di etichette numeriche intere, che corrispondono ai vari cluster.

Nella tabella 4.1 vengono mostrati gli algoritmi di clustering implementati in scikit-learn e citati durante il lavoro di tesi, mostrando la metrica utilizzata per definire la similarità.

Il modulo scikit-learn è stato utilizzato durante il lavoro di tesi perché fornisce l'implementazione di numerosi algoritmi di clustering.

Tabella 4.1: Caratteristiche degli algoritmi di clustering implementati in scikit-learn citati durante il lavoro di tesi

| Algoritmo | Metrica |
|---|---|
| K-means | distanza tra punti |
| Affinity propagation | distanza (grafo) |
| DBSCAN | distanza tra punti più vicini |

### *NetworkX*

Come descritto nell'articolo di Hagberg et al. [68], *NetworkX* [69] è un pacchetto scritto in Python per l'esplorazione e l'analisi di grafi. Il nucleo del tool contiene funzionalità che permettono di:

- rappresentare diverse tipologie di grafo;
- manipolare in maniera semplice i grafi;
- connettersi ad altri moduli di SciPy;
- utilizzare diversi algoritmi per il calcolo delle proprietà del grafo.

NetworkX è stato utilizzato nel lavoro di tesi per la creazione del grafo ricavato dai vettori dei tweet e le distanze tra di loro durante le prove relative agli algoritmi di clustering su grafi.

### *hdbscan*

L'implementazione dell'algoritmo HDBSCAN utilizzata per questo lavoro di tesi è la libreria *hdbscan*, sviluppata da Leland McInnes all'interno della piattaforma GitHub [70], pubblicata in un articolo scritto da lui nel 2017 [71]. L'implementazione di McInnes fa parte della famiglia di pacchetti software integrati con scikit-learn.

Il punto centrale della soluzione di McInnes prevede l'utilizzo di un meccanismo che si occupa di variare il parametro $\varepsilon$ che indica la dimensione dell'$\varepsilon$-vicinato di un punto. Questo permette all'algoritmo di essere più resistente alle scelte dei parametri in input da parte dell'utente.

Il pacchetto hdbscan eredita alcune classi da scikit-learn, dunque mantiene una terminologia ed una modalità di utilizzo coerenti con l'implementazione degli algoritmi di clustering descritti precedentemente. Supporta diversi tipi di formato per i dati in ingresso, e potenzialmente richiede solamente il parametro $MinPts$; potenzialmente, perché è possibile specificarne altri per intervenire direttamente sullo svolgimento dell'algoritmo.

In seguito viene riportato un breve esempio, preso dalla documentazione di hdbscan [72], che dimostra la semplicità di utilizzo dell'algoritmo. Per prima cosa vengono importate le librerie:

```
from sklearn.datasets import make_blobs
import pandas as pd
import hdbscan
```

Vengono poi generati alcuni dati adatti al testing degli algoritmi di clustering:

```
blobs, labels = make_blobs(n_samples=2000, n_features=10)
pd.DataFrame(blobs).head() #mostra in output i dati d'esempio
```

Per effettuare il clustering , è necessario genearare un oggetto apposito, che viene poi utilizzato per assegnare effettivamente le etichette dei cluster ai dati:

```
clusterer = hdbscan.HDBSCAN(min_cluster_size = 10)
clusterer.fit(blobs)
```

L'unico parametro utilizzato per inizializzare l'oggetto è stato `min_cluster_size`, che corrisponde esattamente al parametro $MinPts$ e rappresenta il numero minimo di punti che rendono un $\varepsilon$-vicinato un cluster. La selezione dei parametri è un passaggio delicato per l'utilizzo dell'algoritmo HDBSCAN, dunque viene trattata nel dettaglio alla fine di questo capitolo.

Le etichette dei cluster vengono memorizzate in un array, che indica per ogni elemento dell'input il rispettivo cluster. Tale array è accessibile attraverso un attributo dell'oggetto di clustering, che è `cluster.labels_`. I dati a cui è stata assegnata l'etichetta `-1` sono stati considerati come rumore dall'algoritmo, dunque non devono essere considerati durante la valutazione dei risultati del clustering.

La libreria implementa una modalità di assegnamento del punto al cluster di tipo soft, dunque ad ogni coppia punto-etichetta viene associato un punteggio di decisione che esprime la forza di questo assegnamento. Un punteggio pari a `0.0` indica la non appartenenza al cluster, e viene dato a tutti i punti etichettati come rumore, mentre un punteggio di `1.0` identifica un punto al centro del cluster. I punteggi di decisione possono essere visti utilizzando l'attributo `probabilities_`.

L'implementazione dell'algoritmo fornita da McInnes è stata utilizzata nella fase finale del lavoro di tesi per poter effettuare il clustering dei vettori dei tweet, dunque per individuare i topic.

### Selezione dei parametri

Infine è necessario descrivere nel dettaglio i parametri dell'algoritmo HDBSCAN e la metodologia da adottare per la loro scelta. Tale metodologia viene descritta in una

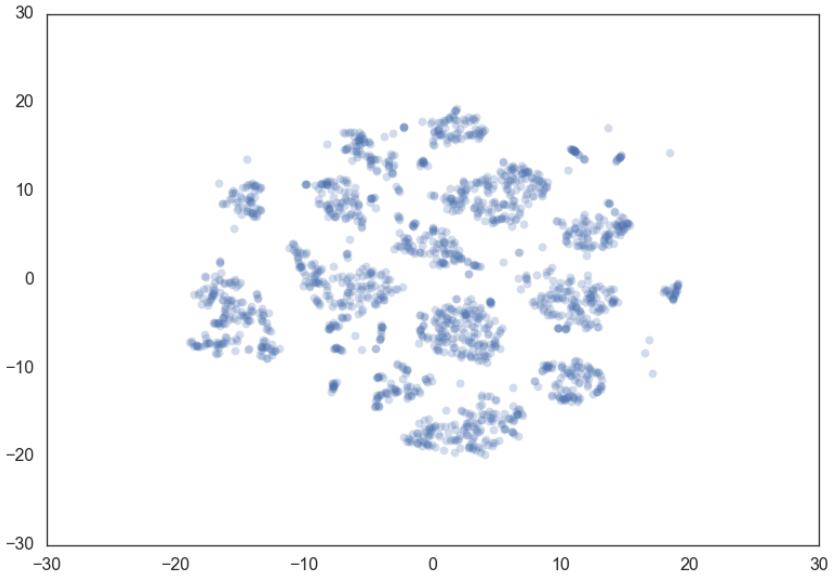


Figura 4.6: Rappresentazione grafica dei punti casuali scelti dall'autore per mostrare l'influenza del valore dei parametri.

pagina dedicata della documentazione fornita da McInnes [73]. Lo stesso McInnes afferma che nonostante la classe `HDBSCAN` abbia un elevato numero di parametri con cui essere inizializzata, pochi di essi influiscono effettivamente sul clustering.

L'autore propone un insieme di punti generati casualmente per meglio mostrare gli effetti della selezione dei valori dei parametri, rappresentati graficamente dalla figura 4.6.

Il primo parametro è `min_cluster_size`, che corrisponde al $MinPts$ descritto durante la presentazione dell'algoritmo nel capitolo del clustering. `min_cluster_size` definisce intuitivamente la dimensione minima che consente ad un raggruppamento di essere chiamato clustering; più formalmente è la dimensione minima per cui l'$\varepsilon$-vicinato di un punto può essere definito cluster.

Nelle figure 4.7 e 4.8 si può notare come il numero dei cluster diminuisca all'aumentare del valore del parametro `min_cluster_size`.

Il grafico invece generato da un clustering con `min_cluster_size = 60`, mostrato nella figura 4.9, presenta un'anomalia: ha infatti meno cluster di quelli attesi. Gli elementi di alcuni cluster che avevano più di 60 elementi sono stati etichettati come rumore.

Questo accade a causa di un secondo parametro, `min_samples`, che, qualora non specificato, assume come valore di default quello di `min_cluster_size`. Questo parametro esprime la misura di quanto il clustering debba essere conservativo: più è alto il valore più sarà conservativo, dunque più punti saranno dichiarati come rumore e più cluster saranno progressivamente ristretti ad aree più dense. Il risultato della riduzione del valore

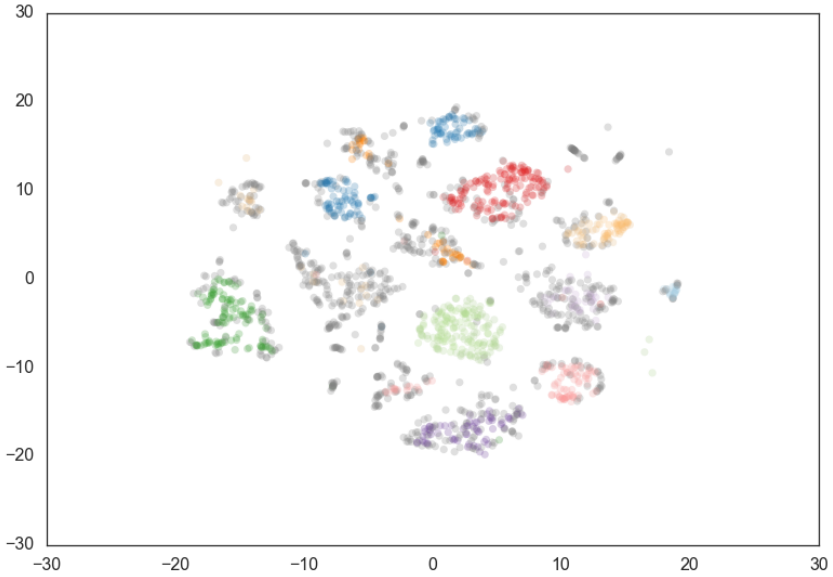


Figura 4.7: Il grafico mostra i cluster con `min_cluster_size = 15`. I punti colorati in grigio rappresentano il rumore.

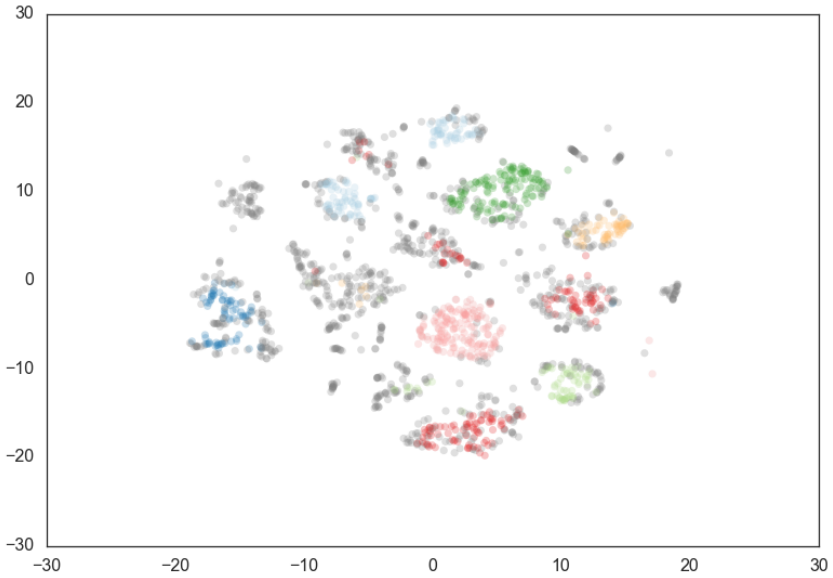


Figura 4.8: Il grafico mostra i cluster con `min_cluster_size = 30`.

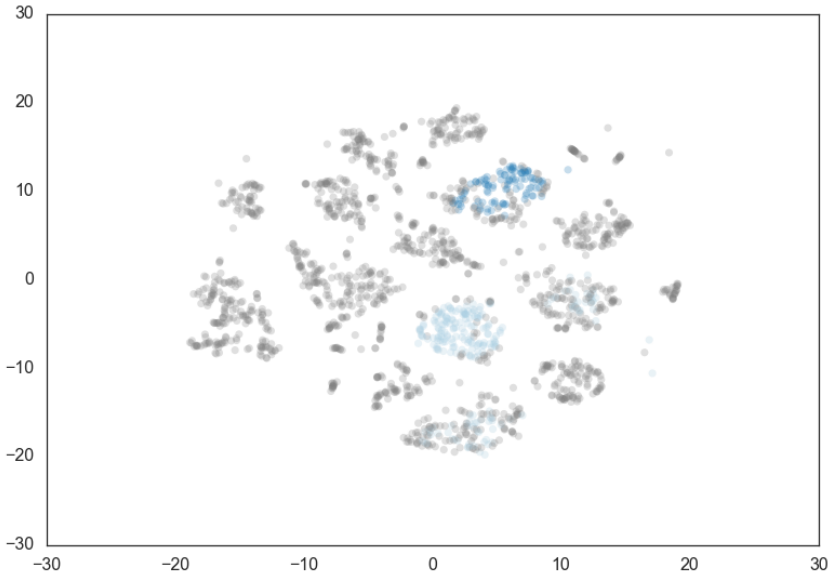


Figura 4.9: Grafico dei cluster generati con `min_cluster_size = 60`.

di `min_samples` può essere visto nelle figure 4.10 e 4.11.

Un altro parametro che influenza i risultati del clustering è `alpha`. L'utilizzo viene fortemente sconsigliato dall'autore, poiché `alpha` determina quanto il clustering debba essere conservativo, ma su una scala più stretta. Per default si ha `alpha = 1`, mentre nella figura 4.12 si è fissato `alpha = 1.3`.

Infine hdbscan supporta un parametro extra, `cluster_selection_method`, che determina come avviene la trasformazione dal clustering gerarchico a quello finale. Senza entrare troppo nel dettaglio, di default il metodo di selezione è *Excess of Mass*, impostato come `cluster_selection_method = 'eom'`: tale metodo tende a generare uno o due grossi cluster circondati da cluster più piccoli. Una scelta per evitare questo problema può essere quella di fissare `cluster_selection_method = 'leaf'`, che permette di ottenere cluster più omogenei.

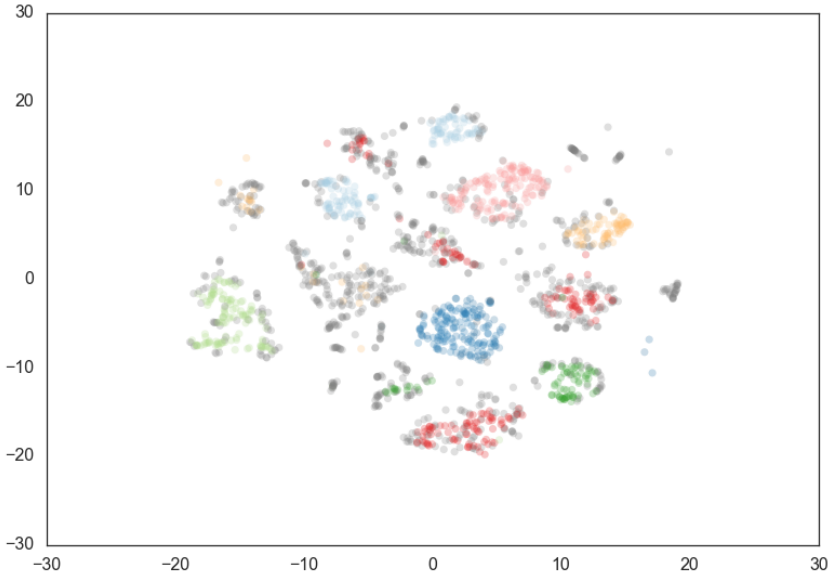


Figura 4.10: Il grafico mostra i cluster con `min_sample = 15`.

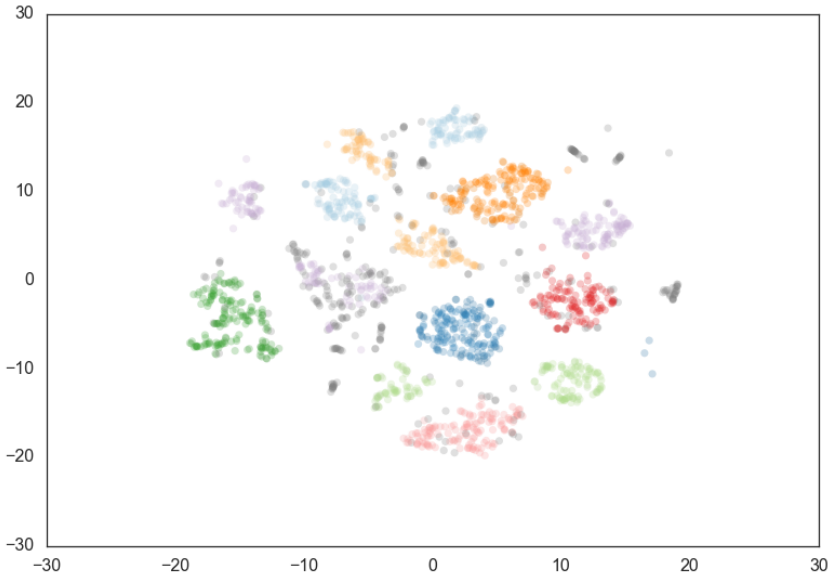


Figura 4.11: Il grafico mostra i cluster con `min_sample = 1`.

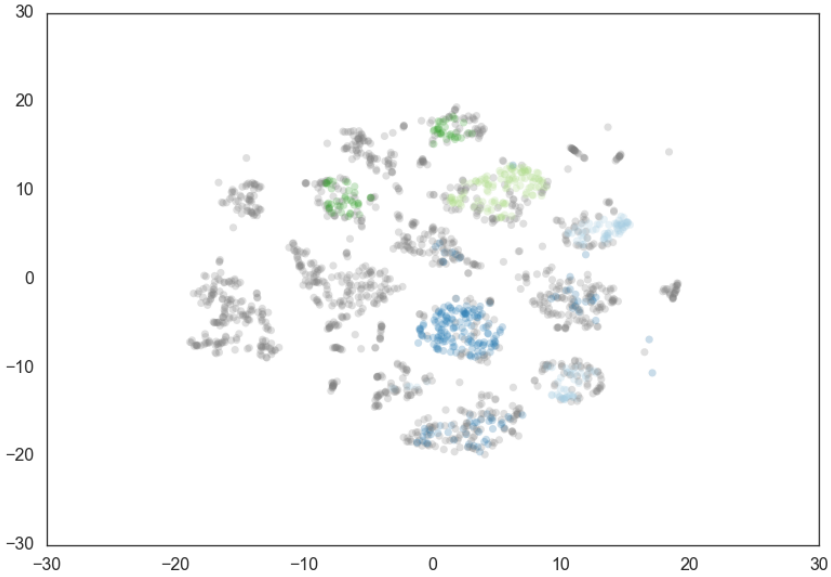


Figura 4.12: Grafico dei cluster generati con `min_cluster_size = 60`, `min_samples = 15`, `alpha = 1.3`.

# Capitolo 5

# Implementazione

In questo capitolo vengono descritti i due script che formano il nucleo della soluzione implementata durante questo lavoro di tesi. Ci si concentrerà dunque sull'analisi del codice presente nello script Python che implementa l'addestramento del modello e di quello che implementa il topic detection.

## 5.1 Addestramento del modello Doc2Vec

Per meglio comprendere le operazioni dello script *topic_detection.py*, descritte nella sezione successiva, è necessario soffermarsi sull'implementazione di un altro script, non incluso direttamente nella soluzione inserita in SpagoBI ma essenziale per il suo funzionamento. Tale script si occupa infati dell'addestramento di un modello Paragraph Vector, con il quale verranno ricavati i vettori dei tweet scaricati da SpagoBI.

Come si è detto nel capitolo che descriveva Paragraph Vector, l'algoritmo sviluppato da Le e Mikolov [26] necessita di due passaggi fondamentali:

- addestramento per creare i vettori delle parole e dei paragrafi/documenti;
- passo di inference, per ricavare i vettori dei paragrafi non ancora visti.

Per poter dunque elaborare i tweet scaricati da SpagoBI è necessario creare un modello che permetta di effettuare un passo di inference dei vettori dei tweet appena scaricati. Questo script si occupa della prima parte della soluzione, cioè quella preposta alla creazione del modello Paragraph Vector in grado di trasformare i tweet in vettori.

Per effettuare l'addestramento del modello l'implementazione proposta utilizza il modulo *Doc2Vec* della libreria Gensim, descritta nel capitolo precedente, poiché contenente le funzionalità dell'algoritmo Paragraph Vector di Le e Mikolov.

Il corpus utilizzato è un insieme di tweet (non etichettati) formato da 2 milioni di tweet.

Il punto centrale dell'addestramento del modello è quello della decisione dei parametri necessari alla sua inizializzazione. Seguendo la documentazione della libreria Gensim [62] offerta dal suo creatore Radim Řehůřek, vengono in seguito descritti tali parametri:

- `dm` definisce l'algoritmo di addestramento del modello tra i due proposti da Le e Mikolov:
  - con `dm = 1` viene utilizzato l'algoritmo PV-DM;
  - altrimenti viene utilizzato l'algoritmo PV-DBOW;
- `dm_mean` definisce l'operazione da utilizzare sui vettori del contesto durante la fase di predizione (utilizzabile solo se `dm = 1`):
  - `dm_mean = 0` indica l'utilizzo della somma tra i vettori;
  - `dm_mean = 1` indica l'utilizzo della media tra i vettori;
- `dm_concat`: definisce l'utilizzo della concatenazione come operazione da utilizzare sui vettori del contesto durante la fase di predizione (utilizzabile solo se `dm = 1`);
- `size` definisce la dimensionalità dei vettori del modello;
- `window` definisce la distanza massima che può intercorrere tra la parola da predire e le parole del contesto all'interno del documento;
- `alpha` definisce il tasso di apprendimento iniziale;
- `sample` definisce una soglia per configurare a quali parole (prese casualmente) ad alta frequenza deve essere ridotta la frequenza;
- `negative`, se posto ad un numero maggiore di zero, definisce l'attivazione del campionamento negativo, per cui il parametro definisce il numero di parole da estrarre e da contare come rumore;
- `min_count` definisce una frequenza; le parole con frequenza inferiore a questa vengono ignorate durante l'addestramento del modello;
- `workers` definisce il numero massimo di *worker thread* che possono essere utilizzati durante l'addestramento del modello.

Un ultimo parametro, che può non essere specificato durante l'inizializzazione del modello, è il numero di iterazioni (*epoch*) che il modello deve effettuare sul corpus durante l'addestramento. Il corpus utilizzato per l'addestramento del modello può infatti essere specificato anche dopo l'inizializzazione del modello.

Per prima cosa dunque lo script inizializza il modello utilizzando i seguenti valori per i parametri:

```
model =  Doc2Vec(
    dm = 1, dm_concat = 1,
    size = 300, window = 5,
    alpha = 0.07, sample = 1e-6,
    negative = 5, min_count = 11,
    workers = multiprocessing.cpu_count()
)
```

Il valore del parametro `workers` viene elaborato dalla libreria `multiprocessing`, in modo da calcolare in maniera dinamica il numero di processori massimo per ogni macchina su cui si effettua la computazione. Vengono dunque tutte le parole con frequenza minore di 11, si ha una window di lunghezza 5, vengono generati dei vettori di dimensione 300 Per addestrare il modello è stato richiesto un tempo di circa 67 minuti.

Lo script dunque, posto `sources` come l'insieme dei documenti (il corpus) utilizzato per addestrare il modello effettua le operazioni necessarie alla creazione del dizionario delle parole:

```
sentences = TaggedLineSentence(sources)
train_array = sentences.train_data()
model.build_vocab(train_array)
```

Queste operazioni servono per manipolare il corpus, affinché possa essere utilizzato dal metodo `build_vocab()` per la creazione del dizionario.

Viene poi svolto un ciclo *for* che conta un numero di iterazioni pari al valore di `epochs` scelto in precedenza, per addestrare il modello sul corpus:

```
for epoch in range epochs:
  model.train(train_data)
model.save('./model_filename.d2v')
```

Il metodo `save()` serve per poter salvare il file contenente il modello direttamente sul disco. Intuitivamente, per caricare il modello dal disco, lo script per il topic detection utilizzerà il metodo `load()`.

Il modello addestrato è stato dunque salvato sul disco, ed è pronto per essere utilizzato dallo script successivo per ricavare i vettori dei tweet scaricati da SpagoBI. Si noti come, una volta addestrato il modello, esso non venga più aggiornato, ma utilizzato come componente a sé stante. Per questo lo script di addestramento del modello non viene

integrato all'interno della soluzione inserita nel modulo di social network analysis di SpagoBI, dove invece andrà il modello da esso generato.

## 5.2 Topic Detection

Questa sezione si concentra sul nucleo vero e proprio della soluzione implementata per il lavoro di tesi, cioè lo script *topic_detection.py*, che si occupa:

- della trasformazione di tweet in vettori utilizzando un'implementazione dell'algoritmo Paragraph Vector;
- della partizione dei vettori dei tweet in cluster o topic utilizzando un'implementazione dell'algoritmo HDBSCAN.

Come già detto nel capitolo precedente, questo script viene chiamato da una modifica dello script R *Topic_Modeling_Twitter_TM_outputDB_unificato.r*.

In seguito verranno descritte le funzionalità presenti nel codice dello script, con particolare attenzione alle parti già anticipate ed a quelle che meglio mostrano le idee alla base dell'implementazione della soluzione.

### 5.2.1 Librerie

Il primo punto del codice da analizzare sono le **librerie** utilizzate per implementare la soluzione. Sono state importate le librerie mostrate nel codice seguente:

```
1 from gensim.models.doc2vec import Doc2Vec
2 import hdbscan
3 import pymysql
4 import re
5 import sys
6 import multiprocessing
7 import numpy as np
```

Tali librerie vengono importate rispettivamente per utilizzare:

1. l'implementazione degli algoritmi Paragraph Vector e l'interazione con il modello addestrato, descritto nella sezione precedente;
2. l'implementazione dell'algoritmo HDBSCAN descritta nel capitolo precedente;
3. le funzionalità di interazione con il database MySQL contenente i tweet scaricati da SpagoBI;

4. le espressioni regolari necessarie alla "pulizia" dei tweet da eventuali simboli, caratteri, parole indesiderate;
5. i parametri presi dal lancio dello script;
6. le funzionalità in grado di fornire il numero di processori presenti sull'hardware che si occupa della computazione;
7. le funzionalità necessarie alla manipolazione dei vettori e degli array.

### 5.2.2 Ottenimento dei tweet

Successivamente si ricava la variabile `search_id`, che identifica la chaive di ricerca dei tweet all'interno del database MySQL utilizzato da SpagoBI. Tale valore viene inserito dallo script R *Topic_Modeling_Twitter_TM_outputDB_unificato.r* per lanciare la parte di topic detection, quindi deve essere catturato come argomento dal codice Python:

```
8  search_id = sys.argv[1]
```

Viene creata poi una connessione al file di log, che contiene nel nome il valore della variabile `search_id`, in modo che l'utente del sistema possa consultarlo per controllare l'andamento del topic detection. Tale file di log viene aggiornato prima e dopo ogni operazione importante effettuata dallo script per il topic detection.

Ottenuto dunque il valore della variabile `search_id` è possibile effettuare una connessione al database MySQL, in particolare alla tabella *TWITTER_DATA*, per poter ricavare i tweet risultati dalla ricerca effettuata dal modulo di social network analysis di SpagoBI.

```
9  connection = pymysql.connect(
10   user='user', password='password',
11   host='127.0.0.1', port=3306, database='dbname'
12 )
13 tweets = []
14 tweet_indexes = []
15 c = connection.cursor()
16 query = (
17   'SELECT TWEET_ID, TWEET_TEXT FROM TWITTER_DATA WHERE SEARCH_ID='
18   + str(search_id)
19 )
20 c.execute(query)
```

Questo codice permette di ottenere all'interno del cursore `c` tutti gli identificativi e i contenuti testuali dei tweet, contenuti nella tabella *TWITTER_DATA*, ricavati da SpagoBI durante la ricerca con valore `search_id`.

Il passo successivo è quello di eliminare tutti i retweet: o, meglio, quei retweet non effettuati tramite una funzionalità di Twitter [43] ma semplicemente copiando e citando il tweet di un'altra persona. Tweet di questo tipo compaiono infatti in alto numero all'interno delle ricerche effettuate su Twitter (sia tramite SpagoBI, sia tramite librerie differenti): basti pensare che, durante lo sviluppo del lavoro di tesi, si incontravano spesso ricerche contenenti anche più del 50% dei tweet di questo tipo.

```
21  for (TWEET_ID, TWEET_TEXT) in c:
22      if "RT" not in TWEET_TEXT:
23        tweets += [TWEET_TEXT]
24        tweet_indexes += [TWEET_ID]
```

Mantenere un array contenente gli identificativi dei tweet verrà utile durante l'aggiornamento finale alla tabella, quando dovranno essere aggiunti i cluster corrispondenti ai tweet.

### 5.2.3 *Text preprocessing*

Conclusa la fase di estrazione dei tweet dal database si hanno due array: uno contenente i tweet estratti, ed uno contenente i rispettivi identificatori. Le parti testuali dei tweet però contengono diversi problemi: frasi piene di parole prove di senso, punteggiatura errata, link a pagine web, mezioni ed altro ancora. Per questo, prima di effettuare le operazioni legate all'algoritmo Paragraph Vector è necessario lavorare su questi tweet, per eliminare le parti considerate inutili alla computazione. Comincia poi la fase del cosiddetto *text preprocessing*.

Il *text preprocessing* è il task per convertire il testo in una sequenza ben definita di unità che abbiano significato dal punto di vista linguistico, come definito da Palmer in un capitolo da lui scritto sull'argomento contenuto nel libro *Handbook of Natural Language Processing* [74]. Il text preprocessing può essere suddiviso in due fasi:

1. *document triage*;
2. *text segmentation*.

Il *document triage* è il processo di conversione di un insieme di file digitali in documenti di testo ben definiti. È formato a sua volta da tre passaggi:

- *charachter encoding identification*, cioè determinare la codifica in cui ogni file è stato scritto e, nel caso di necessità, tradurre in una codifica condivisa i file;
- *language identification*, cioè identificare il linguaggio naturale in cui il documento è stato scritto;

- *text sectioning*, cioè identificare il contenuto utile all'interno di un file, scartando gli elementi indesiderati.

La fase di *text segmentation* si occupa invece di suddividere le componenti, cioè parole e frasi, del testo. Questa fase non è stata necessaria durante l'implementazione del lavoro di tesi, poiché i tweet possono essere già considerati come frasi; suddividerli in parole viene invece fatto in alcuni passaggi del codice per necessità da parte di metodi di avere l'input formattato come lista di parole.

Tornando al document triage, si è analizzato quali fasi fosse necessario implementare. Nel caso del codice che implementa la soluzione proposta:

- la codifica dei tweet è UTF-8 [75], dunque non è necessario implementare un sistema di traduzione per uniformare le codifiche;
- la lingua scelta per effettuare il topic detection è l'italiano, dunque si assume che il sistema recuperi solamente tweet in questa lingua.

I primi due passaggi del document triage non vengono dunque approfonditi, mentre la fase di text sectioning è molto importante per la soluzione proposta: già si sono eliminati tutti i tweet contenenti la stringa "RT" per i motivi già elencati prima, ma è necessario filtrare ulteriormente.

Sono stati definiti due file. Il primo contiene tutti i simboli di punteggiatura indesiderati all'interno dei tweet. Questo perché sui social media (e Twitter non è un eccezione) vengono trovati tweet zeppi di punteggiatura ridondante, in particolare quando si parla dei caratteri " . ", " ! " e " ? ". Il secondo file contiene invece quella che viene chiamata *stopword list*, o lista di stopword. Il concetto di stopword viene definito all'interno di un altro capitolo del libro *Handbook of Natural Language Processing*, scritto da Savoy e Gaussier [76]. Le stopword sono tutte le parole ritenute vuote di significato: dunque, che rappresentano solamente rumore per il sistema. Essendo inutili alla computazione devono essere in qualche modo eliminate.

Si è poi proceduti all'eliminazione degli URL, completamente inutili per il task di topic detection, e i nomi degli utenti utilizzati nelle menzioni [44]: entrambe le eliminazioni sono state effettuate attraverso due espressioni regolari. Osservando il codice:

```
25  pattern_URL = re.compile('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\)
      ,]|(?:%[0-9a-fA-F][0-9a-fA-F]))+')
26  pattern_mention = re.compile('(?:@[\w_]+)')
27  current_tweet = pattern_URL.sub('', current_tweet)
28  current_tweet = pattern_mention.sub('', current_tweet)
```

Si consideri la variabile `current_tweet` come una variabile interna ad un ciclo *for* che ad ogni passo contiene un tweet.

Si è deciso di mantenere questi due elenchi separati dallo script, per poterli estendere, gestire o sostituire senza intervenire sul codice Python: posto che il contenuto di questi due file venga caricato dal codice Python in due array, chiamati `punctuation` e `stopwords`, si effettuano le seguenti operazioni sui tweet:

1. vengono eliminati tutti gli URL tramite l'espressione regolare indicata in precedenza;
2. vengono eliminati tutti i nomi degli utenti utilizzati nelle menzioni;
3. i caratteri del tweet vengono tutti portati in minuscolo, in modo da semplificare le operazioni successive;
4. tutti i caratteri appartenenti all'intersezione tra le lettere di ogni parola di ogni tweet e l'array `punctuation` vengono sostituiti con uno spazio vuoto;
5. tutte le parole appartenenti all'intersezione tra le parole del tweet e l'array `stopwords` vengono sostituite con uno spazio vuoto;
6. vengono eliminati gli spazi ridondanti.

### 5.2.4 Doc2Vec

Arrivato a questo punto il sistema ha un array contenente tutti i tweet, puliti secondo le direttive mostrate nella descrizione relativa al text preprocessing. Ora è necessario ricavare i vettori corrispondenti ai vettori così puliti, utilizzando il modello addestrato con la metodologia mostrata nella sezione precedente.

Viene dunque caricato il modello doc2vec salvato dallo script di addestramento del modello:

```
29  model = Doc2Vec.load('./data/model_filename.d2v')
```

Successivamente si procede all'operazione di inference. La documentazione di Gensim [62] descrive il metodo `infer_vector()` come un metodo che, preso in input un documento (tweet, nel caso di questa implementazione) sotto forma di lista di parole, deduce il vettore corrispondente nel modello addestrato.

```
30 tweets_vectors[i] = [model.infer_vector(tweets[i].split())]
```

Si considerino `tweet_vectors` l'array contenente i vettori dei tweet e `tweets[i]` il tweet attuale all'interno di un ciclo *for*, che viene suddiviso in una lista di parole perché il metodo `infer_vector()` richiede una lista di parole.

### 5.2.5 HDBSCAN

Il lavoro che riguarda il modulo *Doc2Vec* della libreria Gensim si è dunque concluso con la trasformazione in vettori dei tweet scaricati da SpagoBI. Come già detto diverse volte durante la discussione di questo lavoro di tesi, il passo successivo è quello di effettuare il clustering su questi vettori: per farlo è stata scelta l'implementazione dell'algoritmo HDBSCAN offerta da McInnes [71].

La metodologia di selezione dei parametri descritta da McInnes nella documentazione della libreria [73] è già stata descritta nel capitolo precedente, dunque si può mostrare direttamente il codice inserito nell'implementazione:

```
31  clusterer = hdbscan.HDBSCAN(
32      core_dist_n_jobs = multiprocessing.cpu_count(),
33      min_cluster_size = 15, min_samples = 1,
34      cluster_selection_method = 'leaf'
35  )
36  cluster_labels = clusterer.fit_predict(tweets_vectors)
```

Viene dunque inizializzato un oggetto `HDBSCAN` con i parametri descritti nel capitolo precedente. Il parametro `core_dist_n_jobs` permette di impostare il numero di thread utilizzabile durante la fase di computazione delle distanze tra i punti, dunque viene impostato con il metodo `cpu_count()` della libreria `multiprocessing`, esattamente come per l'inizializzazione del modello da addestrare. Il metodo `fit_predict()` restituisce un array di etichette, dunque di valori interi che associati ai tweet ne definiscono il cluster di appartenenza.

L'ultima operazione dell'implementazione sarà quella di aggiornamento della tabella per concludere effettivamente il topic detection.

### 5.2.6 Aggiornamento del database

L'ultima fase dell'implementazione della soluzione presentata da questo lavoro di tesi è quella dell'aggiornamento della tabella *TWITTER_DATA*, all'interno del database utilizzato da SpagoBI. Il campo *TOPICS* di questa tabella viene infatti aggiornato con una query che incapsula l'etichetta del cluster assegnata al tweet dall'algoritmo HDBSCAN, effettuata durante il seguente ciclo *for*:

```
37  for index, tweet, label in zip(tweet_indexes, tweets, cluster_labels):
38      update_query = 'UPDATE TWITTER_DATA SET TOPICS = '
39        + str(label)
40        + ' WHERE TWEET_ID = '
41        + str(index)
42      c.execute(update_query)
43      connection.commit()
```

Concluso questo ciclo la connessione al database viene chiusa e il topic detection può considerarsi effettuato. I risultati del topic detection vengono poi mostrati graficamente nelle tab di SpagoBI.

# Capitolo 6

# Valutazione dei risultati

Questo capitolo, che conclude la discussione del lavoro di tesi, si occupa di descrivere i problemi riscontrati durante le fasi di valutazione del sistema e della metodologia adottata per ovviare ad essi. Infine vengono descritti i risultati di due esperimenti effettuati per comprendere le prestazioni del sistema.

## 6.1 Problemi riscontrati

A causa dei diversi problemi che si sono incontrati durante questa delicata fase che conclude lo sviluppo della soluzione, la valutazione del sistema è stata fatta in maniera qualitativa e non formale. La valutazione formale di un sistema di topic detection come quello presentato necessiterebbe infatti di un corpus annotato, cioè contenente le coppie formate dal tweet e dall'identificativo del topic a cui esso è associato. Un corpus così costruito permette di calcolare, per esempio, i valori di metriche come precision, recall e F-score.

Si ricorda che, come descritto nel capitolo di presentazione dei task del TDT, la precision è la metrica che descrive la frazione di documenti individuati dal sistema che sono effettivamente rilevanti, mentre la recall è la metrica che descrive la frazione dei documenti rilevanti che sono stati correttamente rilevati [8]. F-score è una metrica che combina le due metriche precedenti, calcolandone la media armonica e dunque producendo un risultato compreso tra 0 (risultato peggiore) e 1 (perfezione).

Il corpus annotato richiesto dal calcolo di queste due metriche dovrebbe essere formato da un insieme di tweet in italiano, etichettati manualmente in base al topic di cui parlano. Tali etichette permetterebbero di determinare le quantità da assegnare ai fattori del calcolo delle tre metriche.

Il primo problema riscontrato è stato dunque quello legato al fatto che il task di topic detection, come ripetuto più volte, è non supervisionato: non necessita dunque di informazioni aggiuntive sui dati che devono essere elaborati, come per esempio possono essere le etichette dei topic. Un task non supervisionato permette di lavorare con dati privi di informazioni aggiunte da un essere umano o da un sistema automatico ma rende al contempo estremamente complesso reperire corpora annotati: come descritto nel capitolo relativo all'addestramento del modello tramite l'algoritmo Paragraph Vector, i tweet utilizzati per lo sviluppo della soluzione presentata in questo lavoro di tesi sono circa 2 milioni, tutti privi di informazioni aggiuntive.

Altri task dell'elaborazione del linguaggio naturale, come il sentiment analysis [10], sono invece supervisionati: necessitano dunque di un corpus annotato anche per addestrare il modello, dunque fin dall'inizio dello sviluppo del sistema. Solitamente tali sistemi utilizzano un corpus annotato diviso in due parti: training set, utilizzato per addestrare il modello, e test set, per effettuare la valutazione del sistema. Per eseguire il task e valutare la soluzione è dunque necessario un solo corpus annotato: anche in questo caso non è semplice reperire i corpora adatti, ma ne esistono diversi anche grazie a campagne di valutazione come EVALITA [77], che forniscono ai partecipanti corpora per implementare le soluzioni da presentare.

Il primo problema riscontrato nelle fasi di valutazione del sistema riguarda dunque l'assenza di un corpus annotato, dovuta principalmente al fatto che il topic detection è un task non supervisionato, ma anche alle specificità del problema affrontato da questo lavoro di tesi: sarebbe infatti servito un corpus di tweet in italiano etichettato appositamente per il task di topic detection.

Si poteva pensare di confrontare i risultati del sistema presentato con quelli prodotti da SpagoBI: anche questo si è rivelato essere complesso, per una serie di motivi che verranno elencati nella sezione successiva. Il secondo problema riguarda dunque l'impossibilità di avere un confronto diretto con SpagoBI.

Il terzo problema è fortemente legato al secondo, in quanto si potrebbe pensare di confrontare la soluzione proposta con un'implementazione di LDA (utilizzato da SpagoBI), ma l'algoritmo richiede in input anche il numero di topic, che non può essere conosciuto per la modalità in cui vengono scaricati i tweet. SpagoBI implementa una metodologia per determinare tale numero in maniera automatica, ma si è preferito non procedere in questa direzione: come verrà mostrato nella sezione successiva, SpagoBI verrà abbandonato durante questa fase dello sviluppo della soluzione, a causa dei problemi che verranno elencati in seguito.

Data la presenza di questi tre problemi, la valutazione della soluzione implementata è stata fatta in maniera qualitativa e non formale.

## 6.2 Implementazione del *crawler*

In questa sezione viene descritta una seconda versione dell'implementazione, che non è stata pensata per essere inserita in SpagoBI, ma per essere utilizzata come correzione di alcuni problemi riscontrati (esposti in seguito) e dunque come strumento di valutazione del sistema presentato da questo lavoro di tesi.

SpagoBI presenta infatti una serie di problemi non banali nel momento in cui si desidera valutarne le prestazioni, tra i quali spiccano:

- i tweet scaricati in modalità sia storica che in tempo reale (il download dei tweet viene inspiegabilmente interrotto) non sono sufficienti per valutare le implementazioni di Paragraph Vector e di HDBSCAN;
- circa il 60% di questi pochi tweet contiene la stringa "RT" che, come già indicato in precedenza, evidenzia un tweet copiato ed incollato da un originale, dunque ridondante per il topic detection;
- nonostante si possa specificare una lingua per i tweet da scaricare, vengono spesso scaricati tweet in altre lingue, anche estremamente differenti, che creano confusione durante il clustering;
- SpagoBI impiega un lasso di tempo non giustificato per scaricare i tweet: sia in modalità di ricerca in tempo reale che storica scarica circa 15 tweet al minuto;
- SpagoBI implementa un sistema di preprocessing dei tweet rudimentale, che spesso si dimostra fonte di confusione nell'analisi dei risultati, tra i quali compaiono tweet determinati in base a parole che, nel sistema presentato da questo lavoro di tesi, sono state etichettate come stopword.

Inoltre, anche quando scarica abbastanza tweet da poter effettuare il task di topic detection, il numero di tweet scelti per farlo è estremamente ridotto, tale da rendere impossibile un confronto diretto tra la soluzione proposta e il modulo di social network analysis di SpagoBI. HDBSCAN ha infatti bisogno di un buon numero di tweet per effettuare un clustering valido dei dati.

Lo scarso volume dei tweet ed il lungo tempo richiesto per scaricarli contrasta notevolmente con le prestazioni offerte da un banale *twitter crawler*, cioè un semplice script preposto allo scaricamento di tweet in base ad una serie di parametri specificati.

Questi problemi hanno portato alla necessità di implementare un sistema separato, che utilizzasse gli stessi algoritmi e lo stesso modello della soluzione inserita in SpagoBI, ma che li eseguisse su insiemi differenti di tweet.

È stato dunque implementato un twitter crawler che si occupasse di scaricare i tweet attraverso la libreria *Tweepy*, sviluppata da Joshua Roesslein [78]. Questa libreria permette di scaricare i tweet in tempo reale tramite le *Streaming API* di Twitter [79].

I tweet scaricati sono stati elaborati da un parser *JSON* (per scartare tutte le informazioni non richieste) ed inseriti in un database SQLite. SQLite è un RDBMS, a detta degli sviluppatori, "piccolo, veloce, affidabile" [80], ed è stato utilizzato per la sua flessibilità e semplicità di configurazione. I campi della tabella del database ricalcano quelli della tabella *TWITTER_SEARCH* del database di SpagoBI.

L'architettura di questa versione ridotta della soluzione è molto simile a quella descritta nei capitoli precedenti, con la sola differenza nell'assenza di SpagoBI a collegare i vari elementi e lo scaricamento di tweet in tempo reale e non tramite ricerca storica. I tweet vengono dunque scaricati dal crawler, che si occupa poi di inserirli nel database SQLite.

Per fare un parallelo con i problemi riscontrati in SpagoBI, il crawler proposto:

- finché non viene bloccato continua a scaricare tweet, di essi circa il 60% contiene la stringa "RT";
- scarica dunque circa il 40% di tweet utilizzabile, più che sufficienti per verificare il funzionamento dell'implementazione di Paragraph Vector e di HDBSCAN;
- scarica tweet in italiano, con pochissime eccezioni dovute a lemmi in comune tra diverse lingue;
- scarica circa 1000 tweet al minuto, una prestazione che in termini di tempo surclassa SpagoBI.

I tweet vengono poi estratti da uno script uguale a quello utilizzato per il topic detection nella soluzione inserita in SpagoBI, se non fosse che invece di connettersi ad un database MySQL si connette ad un database SQLite. Tale script si occupa di tutti i passaggi del topic detection, dal preprocessing dei tweet al clustering, in maniera analoga a quella descritta nei capitoli precedenti, per poi scrivere i risultati nel database SQLite.

## 6.3 Valutazione dei risultati

Come si è accennato, la valutazione del sistema è stata fatta in maniera qualitativa, utilizzando i tweet ricavati dal crawler in tempo reale. Nonostante l'elevato numero di tweet scaricati, si è deciso di prendere un campione di 2000 tweet casuali per effettuare la

valutazione del task di topic detection. Questo campione è stato ulteriormente suddiviso in due insiemi: uno contenente 500 tweet, mentre l'altro i restanti 1500. Tale partizione è stata fatta per valutare il funzionamento del sistema a fronte di un cambiamento significativo nella dimensione dell'input.

### 6.3.1 Valutazione del sistema su 500 tweet

Il sistema ha rilevato 5 topic, inserendo in essi 53 tweet tweet e scartando 447 tweet, considerati come rumore. Tale percentuale di tweet ignorati da HDBSCAN è normale per il task del lavoro di tesi: durante tutti i test effettuati, il valore in percentuale di rumore si è sempre attestato intorno a cifre analoghe, dipendendo in minima parte dal valore del parametro `min_samples`, che regola quanto il clustering debba essere conservativo.

Segue l'analisi del topic eisultante alcuni tweet contenuti in esso. Non vengono riportati volutamente in maniera completa perché alcuni di essi sono volgari e dunque inappropriati ad essere inseriti in un lavoro di tesi; altri non risultano interessanti all'analisi del topic detection. Vengono comunque considerati nella valutazione complessiva.

I tweet che vengono chiusi dai punti di sospensione "..." indicano che il tweet supera il numero massimo di caratteri e che è stato condiviso attraverso un altro social network, spesso Facebook. Vengono sempre seguiti dall'URL che collega al testo completo, omesso perché non aggiunge nulla al contenuto del tweet. Sono state inoltre omesse le menzioni. Gli errori di punteggiatura e grammaticali sono stati lasciati dall'originale.

**Topic 0**: il topic etichettato come 0 dal sistema mostra alcuni tweet non eliminati dai vari passaggi di preprocessing che puliscono i tweet da eventuali simboli o parole indesiderate. Non è stato possibile raccogliere tutti i simboli che non hanno valore per il topic detection, ma il sistema dimostra comunque coerenza ad etichettare questi tweet come appartenenti allo stesso cluster. Questo topic dimostra dunque una buona coerenza interna.

**Topic 1**: il topic etichettato come 1 dal sistema contiene 3 tweet. Il primo tweet comunica l'orario di apertura di un convegno, il secondo parla di una notizia che riguarda un personaggio anche televisivo, mentre il terzo parla esplicitamente di un programma televisivo. In complesso il cluster sembra avere come topic principale quello della televisione: l'errore sul primo tweet può essere imputabile ad un'ambiguità legata alla comunicazione di un orario.

- *"Il Torrone Artigianale di Taurianova: scenari e prospettive future" Convegno di apertura, Venerdì 8 ore 17.00*
- *se supportate ancora melanie martinez siete ancora peggio di lei che la difendete non importa se è famosa o se è un ...*

- *E' iniziato IL MIO REGNO PER UN FEGATO - S su #mediasetitalia2*

**Topic 2**: il topic etichettato come 2 dal sistema mostra alcuni difetti evidenti. Gli argomenti trattati, dunque i topic che si possono notare dentro al cluster sono: calcio, videogiochi, politica. L'ultimo tweet si riferisce ad una vaga "finale", che può essere interpretata dal sistema come finale sportiva: più probabilmente si tratta della finale di un noto programma televisivo. L'incoerenza all'interno del topic identificato dal sistema è evidente.

- *#Juventus Amoruso sicuro: "La Juventus resta la squadra da battere, ma l'Inter arriva alla pari" ...*
- *Il mercato avrà cadenza mensile e prevede la vendita di prodotti del territorio con la collaborazione dei ...*
- *Il main theme di #XenobladeChronicles racchiude in sé tutte le emozioni di un'incredibile avventura. Buoni feels!*
- *TG VICENZA (04/12/2017) - PRIMARIE, LA VERSIONE DEL CENTRODESTRA tramite YouTube*
- *Finalmente la finale! Chi ne uscirà vincitore? :)*

**Topic 3**: il topic etichettato come 3 dal sistema presenta incongruenze simili a quelle mostrate dal topic precedente. Non è sufficiente per dire che il topic principale è quello, ma compaiono 2 tweet su 10 che parlano di leggi e giustizia. Anche questo topic dimostra dunque poca coerenza interna, non permettendo di individuare con certezza l'argomento principale. Si notano anche tweet legati alla musica e all'arte.

- *Unisa, domani gli studenti incontrano Federico Buffa nell'ambito di un progetto con la Banca Monte Pruno.*
- *Il problema sono quei genitori che non ci provano nemmeno.*
- *Ripartiamo dalla difesa del suolo, una di quelle battaglie che doveva essere prioritaria in questa legislatura e ...*
- *Non una parola sul fatto che i responsabili, condannati in via definitiva, siano tranquillamente a piede libero in ...*

**Topic 4**: il topic etichettato come 4 dal sistema è quello di dimensione maggiore, e contiene circa metà dei tweet etichettati. Anch'esso presenta tweet vaghi ed incongruenti, in linea con il topic 2 e 3: compaiono diversi riferimenti alla tecnologia, al calcio, alla politica; non è possibile nemmeno in questo caso stabilire con sicurezza quale sia il topic predominante all'interno del cluster.

- *Disponibile il nuovo aggiornamento del sistema 4.1.0 per Nintendo Switch*
- *ma perchè i giocatori del MC dovrebbero rischiare di farsi male in una partita che per loro vale zero? ...*
- *Serve una operazione trasparente abbiamo letto i proclami dell'ex Governo che il bilancio era stato sanato, ...*

Osservando i topic 2, 3 e 4 si comprende come il sistema implementato abbia dei limiti evidenti: non vi sono dei topic semplici da definire e i tweet vengono apparentemente etichettati senza un criterio. Andando a visionare inoltre diversi tweet scartati dal clustering, dunque etichettati come rumore, se ne possono individuare diversi non solo estremamente correlati a temi trattati da quelli clusterizzati dal sistema, ma che utilizzano le stesse parole.

### 6.3.2 Valutazione del sistema su 1500 tweet

Di seguito si mostrano i risultati ottenuti svolgendo il task di topic detection sulla seconda parte dei 2000 tweet, cioè quella composta da 1500 tweet.

Prima di procedere all'analisi, che si svolgerà in maniera analoga a quella utilizzata per la prima porzione di tweet, è però doveroso sottolineare che durante i test, per evitare da una parte l'esplosione del numero di topic, dall'altra per non etichettare tutto come rumore, si è reso necessario modificare i valori dei parametri di inizializzazione di HDBSCAN. Mantenerli infatti fissi ha mostrato qualche limite in termini di flessibilità dell'algoritmo, riscontrabili nel numero di cluster generati e nella quantità di tweet etichettati come rumore. Il risultato che si va a mostrare riflette dunque alcuni cambiamenti rispetto al clustering effettuato su una porzione di tweet di dimensione inferiore.

Il sistema ha rilevato 7 topic, inserendo in essi 140 tweet e scartando 1360 tweet considerandoli come rumore. Si noti come la percentuale di tweet etichettati come rumore dal sistema rispecchi quella mostrata nel clustering sulla porzione precedente di tweet.

**Topic 0**: il topic etichettato come 0 dal sistema contiene diversi tweet vaghi, che probabilmente un annotatore umano avrebbe etichettato come rumore. In questo topic, composto da 11 tweet, troviamo un numero elevato di argomenti apparentemente scollegati tra loro. Il topic dunque non è ben definito, men che meno coerente.

- *Nel cuore della #Gallura la tradizione si rinnova con i #VINI #SARDEGNA IN PUREZZA*
- *Ultimi biglietti disponibili per sentire cantare i Pink Floyd, dalla corista e dal musicista che per anni li ...*

- *Il lisergico carrello italiano dei programmi economici*

**Topic 1**: il topic etichettato come 1 dal sistema mostra gli stessi problemi del topic precedente, cioè tweet assimilabili al rumore e numero elevato di argomenti scollegati.

**Topic 2**: il topic etichettato come 2 dal sistema mostra una maggior coerenza dei precedenti, poiché su 10 tweet più di metà parlano di televisione, cinema, teatro e libri. Pur non essendo lo stesso topic si riscontra una maggiore omogeneità, anche se non sufficiente per valutare corretto il risultato del task di topic detection.

- *Non credete a quelli che vi dicono che si possono guardare certi programmi televisivi ed essere comunque . . .*
- *queste sono perle che vanno condivise con tutto il mondo. #JohnnyDepp*
- *Ballet Nacional Sodre de L' Uruguay!!! A Vicenza il 5 Dicembre . . . N°1.*

**Topic 3**: il topic etichettato come 3 dal sistema mostra tweet che parlano di giornalismo, mostrando una coerenza maggiore degli altri topic, ma, anche in questo caso, non è sufficiente per affermare che il topic rappresentato dal cluster sia effettivamente quello. Compaiono molti tweet che un annotatore umano avrebbe etichettato come rumore.

**Topic 4**: il topic etichettato come 4 dal sistema dimostra una leggera coerenza rispetto al topic sport, ma anche in questo caso i numerosi tweet che trattano di altri argomenti, come cinema, musica, politica, tecnologia e vino, non permettono di definirli come i topic rappresentati dal cluster.

- *"I due punti persi con la Fiorentina bruciano e continueranno a bruciare ancora tanto"*
- *Dopo la telecronaca Pardo/Nava su Fifa non ci può essere di peggio. Nava era meglio . . .*
- *La Delegazione Ais Etruria di Viterbo e l'Associazione a Tavola con Bacco di Perugia in visita a Carpineto, Montepulciano*

**Topic 5**: il topic etichettato come 5 dal sistema si ricolloca nell'insieme di cluster incoerenti e non validi per identificare il topic trattato. Tanti tweet etichettabili come rumore impediscono di effettuare ragionamenti di coerenza.

**Topic 6**: il topic etichettato come 6 dal sistema svolge una funzione analoga al topic 0 dell'esperimento sui 500 tweet, raccogliendo dunque tutti quei tweet che non sono stati eliminati dal preprocessing anche se privi di qualsivoglia contenuto. Questo topic dimostra buona coerenza interna, anche se inutile per giudicare il sistema.

## 6.4 Valutazione complessiva del sistema

In conclusione, per quanto si è potuto osservare in questi due esperimenti, il task di topic detection non è stato effettuato al meglio dal sistema. Le cause possono essere numerose: una plausibile può essere legata la corpus utilizzato per l'addestramento del modello Paragraph Vector. Osservando i tweet scaricati dal crawler, si può osservare come molti parlino di argomenti cosiddetti *di tendenza*, e che spesso includono nomi di persone o parole mai utilizzate in determinati contesti. Twitter è infatti utilizzato per commentare in tempo reale le notizie e gli avvenimenti nel mondo, dunque la maggior parte dei tweet corrisponde ad una porzione di discussione di attualità nella sua più larga accezione. Il corpus utilizzato potrebbe contenere tweet obsoleti, dunque non ottimali, per la generazione dello spazio vettoriale che andrà a mappare invece tweet recenti se non, come in questo caso, che trattano di notizie in tempo reale.

Una seconda causa può essere imputabile ad un'errata scelta dei valori dei parametri dell'algoritmo HDBSCAN, che, come mostrato nel capitolo sul clustering, possono portare a cambiamenti anche drastici nella definizione dei cluster. La scelta dei parametri viene solitamente presa dopo aver valutato formalmente il sistema: misure come precision, recall ed F-score possono dare diverse indicazioni su che direzione prendere per cercare i parametri migliori. Purtroppo, in assenza di un corpus annotato, e di conseguenza di queste misure, sarebbe stato estremamente complesso scegliere accuratamente i valori per i parametri di HDBSCAN. Un possibile sviluppo futuro a questo lavoro di tesi potrebbe essere la ricerca di un sistema in grado di definire (approssimando) i valori dei parametri per HDBSCAN, come verrà descritto nelle conclusioni della discussione del lavoro di tesi.

Il sistema presentato da questo lavoro di tesi non deve però essere visto come un fallimento: i fattori che determinano un cattivo risultato nel topic detection sono diversi e devono essere approfonditi.

Il sistema risolve infatti diversi problemi mostrati da SpagoBI, più legati sicuramente alla qualità ed alla quantità dei tweet utilizzati per il topic detection, ma comunque dannosi durante lo svolgimento del task.

Inoltre questo lavoro di tesi ha posto diversi interrogativi sull'utilizzo degli algoritmi presentati in un contesto moderno in cui non erano ancora stati testati: tali interrogativi possono essere la base per studi futuri che possono dimostrare o meno la bontà della proposta, ma sicuramente contribuiranno a conoscere meglio le tecnologie che si possono utilizzare per la risoluzione di un task moderno ed affascinante come quello di topic detection.

# Conclusioni

In questo lavoro di tesi si è discusso dell'implementazione di una soluzione al problema del topic detection su tweet, da inserire all'interno del software open source SpagoBI per migliorarne le prestazioni; tale soluzione utilizza l'algoritmo Paragraph Vector per addestrare un modello che effettui la trasformazione vettoriale dei tweet, e l'algoritmo HDBSCAN per clusterizzare i vettori risultanti ed effettivamente determinare i topic. È stata poi presentata un'analisi qualitativa dei risultati, che ha mostrato evidenti difficoltà da parte del sistema nello svolgimento del task di topic detection.

Nel primo capitolo è stata fornita una panoramica sul topic detection, contestualizzata come task della TDT. Di quest'ultima si sono descritte le caratteristiche ed i task in cui è suddivisa; sono state inoltre fornite alcune definizioni utili per la comprensione delle tematiche affrontate.

Il secondo capitolo si è concentrato sulla descrizione delle tecnologie utilizzate, anche storicamente, per risolvere il problema del topic detection, partendo dalla presentazione del Vector Space Model ed arrivando alla descrizione del modello Paragraph Vector, scelto per implementare la soluzione al problema risolto da questo lavoro di tesi.

Nel terzo capitolo si sono descritti i due approcci studiati ed utilizzati per effettuare il clustering dei vettori ricavati dal modello Paragraph Vector: il primo legato alla costruzione di un grafo pesato in base alla similarità tra i vettori; il secondo, più classico, legato alla clusterizzazione diretta dei vettori. Si è data maggior rilevanza al secondo approccio, poiché utilizzato per sviluppare la soluzione presentata da questo lavoro di tesi, con l'implementazione dell'algoritmo HDBSCAN.

Nel quarto capitolo è stata presentata l'architettura della soluzione implementata, introducendo i software a cui viene connessa (Twitter e SpagoBI). Sono poi stati descritte le librerie utilizzate per l'implementazione della soluzione, con particolare attenzione a quelle che hanno permesso di utilizzare Paragraph Vector e HDBSCAN.

Nel quinto capitolo sono stati descritti i due script che formano il nucleo della soluzione implementata durante questo lavoro di tesi, cioè lo script che si occupa dell'addestramento del modello Paragraph Vector e lo script che si occupa di effettuare il task di

topic detection vero e proprio. È stata posta particolare attenzione all'analisi del codice sorgente, che viene mostrato nelle sue componenti più interessanti.

Nel sesto capitolo si sono descritte le problematiche riscontrate nella definizione di una metodologia di valutazione della soluzione. Sono stati presentati i difetti notati durante l'utilizzo di SpagoBI ed è stata mostrata una versione della soluzione implementata appositamente per la loro risoluzione, e dunque per permettere una valutazione qualitativa del sistema. È stata infine descritta una metodologia per effettuare tale valutazione, presentando alcuni esempi per mostrare le difficoltà del sistema nello svolgimento del task di topic detection.

Questo lavoro di tesi ha comunque posto diversi interrogativi sui possibili metodi di topic detection su Twitter: non è da considerarsi un fallimento, innanzitutto perché risolve diversi problemi riscontrati in SpagoBI, ma soprattutto perché mostra l'implementazione, ed il conseguente funzionamento, di un approccio non ancora testato per affrontare questo task.

Questo lavoro di tesi vuole può dunque essere considerato come un punto di partenza per eventuali sviluppi futuri, che possono portare ad una maggiore consapevolezza nell'utilizzo di Paragraph Vector e HDBSCAN per risolvere il task di topic detection.

Il primo dei possibili sviluppi futuri deve essere quello della creazione di un corpus annotato per valutare formalmente il sistema, calcolando per esempio i valori di precision e recall. Tale passo è imprescindibile se si desidera utilizzare la soluzione presentata da questo lavoro di tesi come punto di partenza per miglioramenti o affinamenti.

Il secondo dei possibili sviluppi futuri è quello di ideare, come si accennava nel sesto capitolo, una metodologia per determinare automaticamente i valori dei parametri di inizializzazione dell'algoritmo HDBSCAN in base ai dati di input. Tale metodologia dovrebbe tenere conto della dimensione dei dati in ingresso ed assegnare un valore approssimato ma adatto ai parametri `min_cluster_size`, `min_samples` e `cluster_selection_method` utilizzati nella libreria `hdbscan` [73].

Il terzo dei possibili sviluppi futuri è quello di creare una metodologia per estrarre le parole chiave dai documenti clusterizzati, in modo da assegnare un nome comprensibile ai topic, come viene per esempio fatto nel modello LDA [21]. HDBSCAN si limita infatti ad assegnare etichette numeriche intere ai topic, che si rivelano vuote di significato al momento della presentazione e dell'analisi dei risultati.

# Bibliografia


[1] Jonathan G Fiscus and George R Doddington. Topic detection and tracking evaluation overview. In *Topic detection and tracking*, pages 17–31. Springer, 2002.

[2] Andreas M Kaplan and Michael Haenlein. Users of the world, unite! The challenges and opportunities of social media. *Business horizons*, 53(1):59–68, 2010.

[3] We Are Social LTD. We are social. `https://wearesocial.com/`.

[4] We Are Social LTD. Digital in 2017 - global overview. `https://wearesocial.com/special-reports/digital-in-2017-global-overview`, January 2017.

[5] James Allan, Jaime G Carbonell, George Doddington, Jonathan Yamron, and Yiming Yang. Topic detection and tracking pilot study final report. 1998.

[6] Defense Advanced Research Projects Agency. DARPA. `https://www.darpa.mil/`.

[7] Christopher Cieri, David Graff, Mark Liberman, Nii Martey, and Stephanie Strassel. Large, Multilingual, Broadcast News Corpora for Cooperative Research in Topic Detection and Tracking: The TDT-2 and TDT-3 Corpus Efforts. In *LREC*, pages 925–930, June 2000.

[8] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. Modern information retrieval. volume 463, chapter 3, pages 75–76. ACM press New York, 1999.

[9] Engineering. Spagobi site. `https://www.spagobi.org/`.

[10] Bing Liu. Sentiment Analysis and Opinion Mining. pages 1–2. Morgan & Claypool, 2012.

[11] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[12] Christopher D Manning, Hinrich Schütze, et al. Foundations of statistical natural language processing. volume 999, chapter 15. MIT Press, 1999.

[13] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. Modern information retrieval. volume 463, chapter 3, pages 1–2. ACM press New York, 1999.

[14] Alessandro Lenci. Distributional semantics in linguistic and cognitive research. 20, 01 2008.

[15] Jaime Arguello. Vector space model. *Information Retrieval September*, 25, 2013.

[16] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[17] Gerard Salton and M. J. McGill. *Introduction to modern information retrieval*. McGraw-Hill, 1983.

[18] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391, 1990.

[19] Thomas Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 289–296. Morgan Kaufmann Publishers Inc., 1999.

[20] Steffen Lauritzen. Exchangeability and de Finetti's Theorem. `http://www.stats.ox.ac.uk/~steffen/teaching/grad/definetti.pdf`.

[21] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet Allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[22] Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In *ACL (1)*, pages 238–247, 2014.

[23] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[24] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[25] Frank Robert Palmer. Selected papers of J. R. Firth. pages 238–247. Indiana University Press, 1968.

[26] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1188–1196, 2014.

[27] Christopher D Manning, Hinrich Schütze, et al. Foundations of statistical natural language processing. volume 999, chapter 14. MIT Press, 1999.

[28] Peter J Rousseeuw and L Kaufman. *Finding Groups in Data*. Wiley Online Library, 1990.

[29] Satu Elisa Schaeffer. Graph clustering. *Computer science review*, 1(1):27–64, 2007.

[30] Thomas H. Cormen et al. Introduzione agli algoritmi e strutture dati. pages 970–974. McGraw-Hill, third edition, 2009.

[31] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[32] scikit-learn developers. Affinity propagation. `http://scikit-learn.org/stable/modules/clustering.html#affinity-propagation`.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[34] Axel-Cyrille Ngonga Ngomo and Frank Schumacher. Borderflow: A local graph clustering algorithm for natural language processing. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 547–558. Springer, 2009.

[35] Gary William Flake, Steve Lawrence, and C Lee Giles. Efficient identification of web communities. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 150–160. ACM, 2000.

[36] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[37] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[38] Lingjuan Li and Yang Xi. Research on clustering algorithm and its parallelization strategy. In *Computational and Information Sciences (ICCIS), 2011 International Conference on*, pages 325–328. IEEE, 2011.

[39] Zhiwei Sun. A hierarchical clustering algorithm based on density for data stratification. In *Systems and Informatics (ICSAI), 2012 International Conference on*, pages 2208–2211. IEEE, 2012.

[40] Oracle. Mysql. https://www.mysql.com/it/.

[41] Wikipedia. Twitter. https://it.wikipedia.org/wiki/Twitter.

[42] Ikuhiro Ihara. Our discovery of cramming. https://blog.twitter.com/engineering/en_us/topics/insights/2017/Our-Discovery-of-Cramming.html.

[43] Twitter Inc. Domande frequenti sui retweet. https://support.twitter.com/articles/20172627.

[44] Twitter Inc. Tipi di tweet e dove vengono visualizzati. https://support.twitter.com/articles/249994.

[45] Twitter Inc. Cosa sono gli hashtag? https://support.twitter.com/articles/253564.

[46] Twitter Inc. Domande frequenti sul following. https://support.twitter.com/articles/255335.

[47] I. Iennaco, L. Pernigotti, and S. Scamuzzo. Sentiment analysis with SpagoBI. In *Sentiment Analysis in Social Networks*, pages 189–192. Morgan Kaufmann, 2016.

[48] Engineering. Engineering website. http://www.eng.it/.

[49] Mozilla Foundation. Mozilla public license 2.0. https://www.mozilla.org/en-US/MPL/2.0/.

[50] Engineering. Knowage. https://www.knowage-suite.com/site/home/.

[51] The R Foundation. What is R? https://www.r-project.org/about.html.

[52] Twitter Inc. Twitter application management. https://apps.twitter.com/.

[53] Engineering. Spagobi social analysis. http://wiki.spagobi.org/xwiki/bin/view/spagobi_server/social_analysis.

[54] Python Software Foundation. What's new in Python 3.6. https://docs.python.org/3/whatsnew/3.6.html.

[55] Python Software Foundation. Python 3.6.1. https://www.python.org/downloads/release/python-361/.

[56] Python Software Foundation. Python 3.6.0. https://www.python.org/downloads/release/python-360/.

[57] Travis E Oliphant. Python for scientific computing. *Computing in Science & Engineering*, 9(3), 2007.

[58] Ivan Idris. Python data analysis. Packt Publishing Ltd, 2014.

[59] XMind Ltd. Python for big data. https://www.xmind.net/m/WvfC.

[60] Radim Řehůřek and Petr Sojka. Gensim - statistical semantics in Python. 2011.

[61] Radim Řehůřek and Petr Sojka. Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Citeseer, 2010.

[62] Radim Řehůřek. models.doc2vec – deep learning with paragraph2vec. https://radimrehurek.com/gensim/models/doc2vec.html.

[63] Radim Řehůřek. Doc2vec tutorial. https://rare-technologies.com/doc2vec-tutorial/.

[64] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python. http://www.scipy.org/, 2001.

[65] NumPy developers. Numpy. http://www.numpy.org/.

[66] SciPy developers. Numpy and scipy documentation. https://docs.scipy.org/doc/.

[67] scikit-learn developers. Clustering. http://scikit-learn.org/stable/modules/clustering.html.

[68] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Laboratory (LANL), 2008.

[69] NetworkX developers. Software for complex networks. https://networkx.github.io/.

[70] Leland McInnes. HDBSCAN. https://github.com/scikit-learn-contrib/hdbscan.

[71] Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *The Journal of Open Source Software*, 2(11), mar 2017.

[72] Steve Astels Leland McInnes, John Healy. The hdbscan clustering library. https://hdbscan.readthedocs.io/en/latest/index.html.

[73] Steve Astels Leland McInnes, John Healy. Parameter selection for hdbscan. https://hdbscan.readthedocs.io/en/latest/parameter_selection.html.

[74] David D. Palmer. Text preprocessing. In *Handbook of Natural Language Processing*, pages 9–30. CRC Press, 2010.

[75] F. Yergeau. UTF-8, a transformation format of ISO 10646. https://tools.ietf.org/html/rfc3629.

[76] Jacques Savoy and Eric Gaussier. Information retrieval. In *Handbook of Natural Language Processing*, page 458. CRC Press, 2010.

[77] The EVALITA Community. EVALITA. http://www.evalita.it/.

[78] Joshua Roesslein. Tweepy. http://docs.tweepy.org/en/latest/index.html.

[79] Twitter Inc. Filter realtime tweets. https://developer.twitter.com/en/docs/tweets/filter-realtime/overview.

[80] Dwayne Richard Hipp. SQLite. https://www.sqlite.org/index.html.