A Sabrina

Indice

Introduzione

Questo lavoro di tesi si propone lo scopo di progettare ed implementare un sistema distribuito in grado di acquisire, processare ed estrarre i nuclei informativi di file di testo provenienti dalla rete. Particolare cura è stata dedicata alla costruzione dell'algoritmo e alla scelta del caso di studio: nel primo, in modo particolare, si fondono diversi strumenti, tutti Open Source e tutti accomunati dallo stesso linguaggio e ambiente di programmazione, che consentono sia di gestire un gran numero di dati e sia, soprattuto, di eseguire una parte computazionalmente rilevante del codice in modo distribuito, permettendo un abbattimento notevole dei tempi di esecuzione.

I temi trattati in questo elaborato sono molteplici, ma possiamo suddividerli in due macro-aree: in primo luogo ci si pone il problema di estrarre l'informazione contenuta in una collezione di file di testo, sviluppando metodi e modelli statistici per l'estrapolazione dei termini e, verosimilmente, dei concetti chiave; in secondo luogo, si affronta la questione di gestire grosse moli di dati, architettare procedure di analisi testuale efficienti con gli strumenti a disposizione e rendere l'intero processo sufficientemente veloce e automatizzato da poter essere generalizzato e suscettibile di future applicazioni.

A culmine di tutto il lavoro viene presentato un caso di studio che permette di testare l'intera procedura costruita; in particolare, i file di testo analizzati sono costituiti da articoli di una selezione di testate giornalistiche. L'obiettivo è quello di classificarne i contenuti sulla base dei temi maggiormente trattati durante il periodo di raccolta dati.

Il dettaglio degli argomenti presentati è il seguente.

Nel Capitolo 1 si fornisce una visione generale del problema affrontato. In esso non mancano riferimenti e indicazioni per generalizzazioni e possibili applicazioni. Si cerca inoltre di contestualizzare la procedura, sottolineando la necessità di analisi efficienti in una realtà dove la produzione dei dati è così elevata che la necessità di coadiuvarsi con il calcolatore è ormai un fatto scontato.

Nel Capitolo 2 si analizza l'apparato software utilizzato durante l'intero studio, se ne specificano le funzionalità e le motivazioni di utilizzo. Particolare enfasi si pone sulla questione dell'unitarietà dell'intera procedura,

garantita dall'uso di un solo linguaggio di programmazione che opera a tutti i livelli, e sulla gestione parallela di alcuni compiti da parte dell'algoritmo

Nel Capitolo 3 vengono presentati i principali modelli: di essi si analizzano i dettagli tecnici, si dànno cenni degli algoritmi di implementazione e si fanno, naturalmente, confronti critici per comprendere i punti di forza di ciascuno.

Nel Capitolo 4 si dànno dettagli circa la costruzione dell'intera procedura. In esso si manifesta il contributo originale della tesi: la fusione delle due principali metodologie studiate e l'implementazione di un algoritmo distribuito.

Nel Capitolo 5, infine, si mostrano i risultati dell'analisi compiuta nel caso particolare delle testate giornalistiche: si esegue il codice su una selezione di notizie provenienti da alcuni siti di quotidiani prescelti.

Nell'Appendice A si parlerà dell'acquisizione dei dati provenienti dai social network e si darà il dettaglio della procedura nel caso particolare di Facebook.

Nell'Appendice B sono richiamati i principali concetti di probabilità e statistica strumentali alla comprensione della teoria affrontata.

I codici usati sono interamente disponibili in rete, nella repository Github all'indirizzo:

$$https://github.com/DavideGambocci/Social-Media-Analysis$$

Nota: talvolta gli output sono stati modificati per facilitarne l'impaginazione.

Capitolo 1

Il contesto del problema trattato: tematiche e motivazioni

Secondo uno studio dell'Università di Berkeley [Lym03], nel 2003 sono stati prodotti "... *25 TB*[1] *di file di testo relativi ai giornali, 10 TB relativi alle riviste in genere ... e 195 TB di documenti d'ufficio. Si stima che siano state inviate 610 miliardi di e-mail, per un totale di 11000 T*B "; nel 2010 il solo numero di mail è salito alla enorme cifra di 107×10^{12} secondo [Roe12] e, venendo a dati più recenti, si stima che nel 2017 la media di e-mail inviate sia stata di 269 miliardi al giorno [Rad17]. Nel 2003, inoltre, non era ancora affermata la realtà social, per cui nel conteggio dell'informazione prodotta oggi dobbiamo aggiungere anche file testuali di nuova generazione come *post*, *commenti* e *tweet*: di questi ultimi, nel 2017, ne sono stati prodotti circa 300 miliardi ([IntLS]).

Insomma la crescita dei documenti testuali è stata esponenziale a tutti i livelli: ciò solleva diverse riflessioni e questioni a proposito di tematiche importanti ed attuali. Con l'obiettivo di voler implementare un algoritmo in grado di eseguire tutte le fasi dell'analisi testuale di file provenienti dalla rete (dall'estrazione dei dati all'applicazione di topic model), dobbiamo avere una panoramica di tali questioni e comprendere quali sono le possibili applicazioni e i limiti della procedura che costruiremo. Innanzitutto, è evidente che una così massiccia produzione di dati, può contenere informazioni di indubbio valore per diverse tipologie di analisi, da quelle di mercato a indagini di tipo sociologico. Un altro punto interessante è l'immediatezza con cui questi dati sono reperibili: in passato, per molte analisi, raccogliere i dati necessari poteva essere un'operazione che comportava lunghe attese.

Chiaramente, e lo sottolineeremo anche in seguito, non è detto che tutti i dati siano di ottima qualità, anche se, spesso, ciò dipende anche dal tipo

[1] 1 Terabyte = 1000 Gigabyte

di analisi che si vuole condurre. Un tale flusso di dati, consente in molti casi analisi accurate anche se non risolve certamente tutti i problemi, al contrario, l'aspetto interessante è che ne genera di nuovi, legati soprattutto alla loro gestione e al loro *processing*.

1.1 I *big data* e l'analisi testuale

Dati del genere appena esaminato, visto l'enorme volume che li caratterizza, sono stati definiti con l'aggettivo *big*: i *big data* rappresentano un punto di svolta nell'analisi dei dati ed è doveroso capire come gestirli, pulirli e processarli. Il confronto con la realtà dei *big data* avviene, chiaramente, nella fase di estrazione delle informazioni dalla rete, una fase che possiamo definire di raccolta delle informazioni. Riconosciamo subito che, se da un lato un tale flusso di dati non può essere ignorato perché potenzialmente carico di informazioni interessanti, dall'altro, per un essere umano singolo è impensabile portare a termine un'analisi efficiente in tempi accettabili: il sostegno del calcolatore è necessario, motivo per cui concetti rilevanti di programmazione saranno comunque toccati. Vediamo nel dettaglio con quali sono le questioni principali da risolvere.

È facile intuire quali siano le problematiche legate ai big data: prima di tutto si presenta una difficoltà di gestione: si ha necessità di tecnologie in grado sia di memorizzare i dati in formati adeguati e sia in modalità che ne consentano un'elaborazione computazionalmente sostenibile. La soluzione consiste nella scelta accurata di un database, che, eventualmente, abbia caratteristiche che consentano di gestire efficacemente la mole di informazioni in modo rapido e sostenibile (cfr. Cap. 3).

Le difficoltà computazionali, tuttavia, non si esauriscono esclusivamente nella fase gestionale: esse si palesano anche nell'applicazione dell'algoritmo operativo che deve essere strutturato in modo da gestire l'esecuzione nel modo più efficiente possibile. Sembrerebbe che la crescita della tecnologia hardware, che permette di avere a disposizione piattaforme con specifiche tecniche sempre più avanzate, rappresenti la soluzione al problema, ma ci si convince facilmente del contrario osservando che l'avanzamento tecnologico va di pari passo con l'aumento di produzione dei dati, anzi in molti casi quest'ultima prevale sulla prima che dunque, da sola, non è sufficiente. Occorre *parallelizzare* l'esecuzione: questo vuol dire strutturare il codice in modo che esegua diverse operazioni contemporaneamente (magari interessando diversi core di un processore) con conseguente riduzione dei tempi dovuta alla suddivisione dei compiti eseguiti.

Un'ultima importante questione legata all'aspetto più squisitamente analitico: i *big data* rappresentano una potenziale miniera di informazioni, tuttavia spesso possono portare a conclusioni inevitabilmente distorte, qualora non vengano usati con criterio. Di questa importante tematica non ci occu-

peremo direttamente e l'abbiamo enunciata solamente per completezza. Per un esempio interessante di uso distorto dei *big data* e per utili strategie di pianificazione, si può consultare [Con12].

Una volta conclusa l'operazione di raccolta, occorre operare il cosiddetto *information retrieval*, ovvero il recupero delle informazioni. Durante questa fase si estrae dai dati in memoria, una quantità più piccola di dati rilevanti ai fini dell'analisi finale. Ad esempio, è possibile che durante la memorizzazione dei file si accumuli del codice sorgente della pagina che ospitava il contenuto: tale codice può e deve, naturalmente essere eliminato.

I dati che analizzeremo saranno, ribadiamo, file testuali: l'informazione che da essi vogliamo estrarre rappresenta una sintesi, in qualche senso, del loro contenuto. Ovviamente in tali documenti, l'informazione sarà veicolata mediante uno specifico linguaggio umano, il quale deve essere poi processato dal computer. Si intuisce chiaramente l'insorgenza di un problema molto spinoso: quello di consentire alla macchina di processare tale linguaggio.

1.2 Il linguaggio naturale

È generalmente facile per un essere umano comprendere il contenuto di un testo, rilevarne il concetti principali e sintetizzarlo o rielaborarlo. Per una macchina l'intero processo è più lontano dalle sue funzioni: occorre istruirla alla comprensione del *linguaggio naturale* e alla sua elaborazione, definita anche *Natural Language Processing* (NLP). L'NLP viene suddiviso in diversi sottoprocessi che simulano i livelli di apprendimento umano, i principali (per ulteriori approfondimenti si veda [Zha16]) sono i seguenti

- **Analisi lessicale**: mediante la quale si scompone il discorso nei suoi costituenti principali (*tokenizzazione*) e si assegna loro un ruolo. Per l'italiano, la componente atomica di una frase è costituita dalla parola; in generale, per lingue come il Cinese o le lingue agglutinanti è più difficile scomporre il discorso in quanto non sempre esiste una separazione netta per le parole;
- **Analisi sintattica**: che si occupa di stabilire la funzione logica di una parola all'interno della frase;
- **Analisi semantica**: che consiste nella comprensione del significato proprio di ciascuna parola, anche in relazione alla radice etimologica e al contesto in cui essa si trova. L'obiettivo è quello di comprendere il significato dell'intero discorso analizzato. Specialmente in questa fase si avvertono maggiormente le ambiguità che una lingua presenta e, in contesti molto estremi come le interpretazioni di scritti sacri o antichi, può essere un compito difficile anche per l'essere umano.

Prendiamo come esempio la seguente frase

Il bambino gioca con un giocattolo

Nell'ottica dell'analisi lessicale la frase si scomporrebbe nel modo seguente

$$\underbrace{Il}_{art.}\ \underbrace{bambino}_{nome}\ \underbrace{gioca}_{verbo}\ \underbrace{con}_{prep.}\ \underbrace{un}_{art.}\ \underbrace{giocattolo}_{nome};$$

mediante un'analisi sintattica invece si avrebbe

$$\underbrace{Il\,bambino}_{soggetto}\ \underbrace{gioca}_{predicato}\ \underbrace{con\,un\,giocattolo}_{complemento}.$$

Con l'analisi semantica, invece, si scoprirebbe che *gioco* e*giocattolo* hanno la stessa radice e lo stesso significato e che, probabilmente, esso è il tema centrale della frase.

In generale, ogni analisi aumenta la comprensione del discorso e più livelli si esaminano, maggiore sarà la precisione della macchina nell'individuare il significato del testo che le si chiede di analizzare. Per ciascun sottoprocesso sarebbe opportuno costruire un insieme di strumenti in grado di consentire l'analisi a quel livello. Generalmente, anche se non il solo, lo strumento maggiormente utilizzato è il *dizionario*: la creazione di dizionari appositi consente di comprendere se una parola rappresenta un nome piuttosto che un aggettivo, se possiede un significato affine ad un altro termine in una frase e se è parte di un'espressione idiomatica o di una locuzione. L'assenza di dizionari completi è un problema notevole, vedremo nel prosieguo come cercare di aggirarlo o di impostare una soluzione sub-ottimale per proseguire l'analisi.

La fase successiva è rappresentata dalla capacità del computer di elaborare queste informazioni, estraendone, eventualmente, una sintesi: tale fase è chiamata *text mining*, ovvero la ricerca dei principali gruppi tematici all'interno di file di testo.

Allo stato attuale non è possibile insegnare a un computer a comprendere profondamente un testo, percepire l'ironia o trarre conclusioni critiche, tuttavia, con l'ausilio di alcuni rudimenti di NLP, della probabilità e della statistica, la macchina riesce a riconoscere i termini di maggiore rilevanza, diversi costrutti latenti e strutture più complesse come la sinonimia.

In qualche senso, occorre immaginare in che maniera un computer, che non comprende il linguaggio naturale, possa comunque riuscire a trattarlo. Il modo più semplice è quello di considerare come indicativa solo l'occorrenza delle parole nel testo: i concetti presenti si palesano nei nomi, verbi e aggettivi utilizzati nel discorso e dunque è tra le occorrenze di questi ultimi che bisogna scorgere la sintesi dei contenuti. In generale, intuitivamente, più una parola sarà ripetuta e più è plausibile che essa sarà rilevante ai fini della comprensione di quel testo. Questo approccio è ancora molto grezzo: articoli e congiunzioni ad esempio si ripetono spesso con notevole frequenza

all'interno di un discorso, eppure la loro funzione non ha niente a che fare con l'argomento di cui si sta parlando. Occorre dunque limitare l'analisi solamente ad una categoria selezionata di parole ed è quello che si vedrà in seguito. Quest'idea, seppure molto semplice, di considerare unicamente le parole e la loro frequenza come veicoli del concetto è esattamente la medesima che sta alla base dei modelli che useremo. Due concetti importanti si possono già evidenziare: in questo approccio il ruolo principale spetta alle parole in quanto tali e l'*ordine* con cui esse sono trascritte è perfettamente trascurabile. Un algoritmo che segue una simile linea di pensiero, si dice essere di tipo *bag-of-words*. Il *Latent Semantic Indexing* possiede esattamente tutte le caratteristiche appena enunciate: le sue specifiche tecniche saranno analizzate nel terzo capitolo. È importante, invece, capire come un siffatto approccio si possa migliorare, cercando di cogliere qualche aspetto più complesso che un metodo così concettualmente semplice non è in grado di cogliere. Ci si accorge subito che tale modello poggia su presupposti non sempre verificati: non sempre, infatti, le parole più diffuse sono rappresentative dei concetti centrali di un testo, alla comprensione dei quali noi effettivamente miriamo. Per oltrepassare questo limite, si ragiona in questo modo: supponiamo, per un testo, l'esistenza di alcuni concetti *latenti* (detti *topic*). A partire da essi si può generare la distribuzione delle restanti parole nel testo, sicché i *topic* rappresentano gli archetipi generativi della forma finale del documento analizzato. Tale idea è alla base del *Latent Dirichlet Allocation* che, oltre ad essere anch'esso un modello di tipo *bag-of-word*, rileva l'aspetto distribuzionale, e dunque statistico, insito nella composizione di un testo.

Un modello come il *Latent Semantic Indexing* o come il *Latent Dirichlet Allocation*, in quanto metodo per l'estrazione di contenuto informativo presente in un testo, si definisce *topic model* o modello di *topic extraction*.

I *topic model* sono dunque gli strumenti matematico-statistici che abbiamo a disposizione per lo scopo che ci proponiamo di raggiungere.

1.3 La procedura e la scelta del caso di studio

A questo punto, dopo aver esaminato il contesto che giustifica l'implementazione della nostra procedura, risultano delineati tre aspetti centrali del nostro studio: i *big data*, il processing del linguaggio naturale e i *topic model*. Si comincia a comprendere quali possano essere i requisiti dell'algoritmo da implementare. Innanzitutto si richiede che esso sia in grado di reperire i file testuali dalla rete, di memorizzarli e gestirli e poi di ripulirli dal superfluo di cui inevitabilmente sono carichi. Successivamente, occorre analizzare i file così processati in modo che il computer comprenda quali sono le parti fondamentali del discorso e se ci sono affinità di significato tra i termini incontrati.

Infine, si applicano i modelli di *topic extraction*. Resta da motivare la scelta del caso di studio per testare l'algoritmo.

I contenuti testuali presenti in rete, come abbiamo visto, sono moltissimi e disparati e vanno dalle mail ai tweet, dai blog ai siti istituzionali. In particolare, ci sembra rilevante soffermarci sul settore dell'editoria giornalistica, i cui contenuti sono oggigiorno quasi interamente fruibili in rete. Tale settore rappresenta una parte molto importante della categoria più generale dei *social media*. Con quest'ultima locuzione si intende quel complesso di tecnologie e pratiche che le persone adottano per lo scambio di informazioni in Internet ed estende alla rete quella dimensione una volta esclusiva dei *mass media*. In questo senso il giornalismo rappresenta una forma di contatto tra questi due aspetti, tuttavia le motivazioni della nostra scelta vanno ben oltre. Innanzitutto, il giornalismo produce articoli ad un ritmo costante, il che è fondamentale per le rilevazioni: nel caso di tweet o mail, alcuni specifici contenuti di interesse possono essere postati in modo occasionale e dunque non è scontato che si riesca ad accumulare un sufficiente numero di dati in tempo fissato; gli articoli possiedono, per di più, un'estensione non ridotta e che si presta bene alle analisi dei topic model. A differenza delle pagine sui social o dei blog, inoltre, i giornali appaiono come entità più solide: accade di frequente che i blog vengano presi di mira da attacchi informatici o che le pagine dei social chiudano spontaneamente o cambino, mentre è rarissimo che un giornale di punto in bianco smetta di esistere; a differenza di questi ultimi, inoltre, gli articoli di giornale rappresentano modelli testuali di buona qualità: non si incontrano generalmente errori ortografici e non usano forme di scrittura insolite (nei commenti social, ad esempio è facile imbattersi in espressioni come *xke* invece di *perché*, ecc.). Per di più, le conclusioni che possiamo trarre dall'analisi degli articoli di giornale hanno, potenzialmente, un interesse di pubblico rilievo e sono suscettibili di diverse e interessanti generalizzazioni.

Nel caso esaminato, ci preoccuperemo di enucleare gli argomenti principali di ciascuna testata giornalistica analizzata, allo scopo di classificarla sulla base degli argomenti trattati.

Come più volte sottolineato, l'algoritmo si può adattare anche ad altri contesti nel campo dei social media: basta, ad esempio, modificare la sorgente dei dati acquisiti (nell'appendice finale si vedrà come estrarre anche dati dai social). Gli studi che si possono condurre sono svariati, con applicazioni che vanno dalla *sentiment analysis* alle ricerche in campo sociologico. Esempi di applicazioni esistenti ne abbiamo già forniti.

Un esempio rilevante di applicazioni già sviluppate che devono confrontarsi con l'enorme flusso di dati provenienti dalla rete e con la necessità di classificarli, è costituito dai *motori di ricerca*. Essi devono costantemente analizzare dati voluminosi dalla rete per renderli disponibili all'utenza. L'uso dei topic model in quest'ambito (specialmente quelli che verranno trattati in seguito) è estremamente diffuso.

La nostra procedura, tuttavia, opera diversamente da un motore di ricerca in quanto non si occupa di tabulare i dati per renderli facilmente cercabili, ma si preoccupa di voler estrarre i temi centrali da un testo.

Anche applicazioni specifiche come *Google News* ([GooN]), sebbene partano dai nostri stessi presupposti, funzionano da aggregatori di notizie, mentre lo scopo nostro è quello di analizzarne i contenuti e non solamente di confrontarli. Inoltre il tipo di dato analizzabile, con opportune generalizzazioni (come quella mostrata in appendice), non è necessariamente costituito da articoli di giornale.

Insomma, la ricerca in questo ambito è molto attiva, il tema dell'analisi dei dati dalla rete è molto sentito e speriamo, con i nostri accorgimenti, di dare anche noi un piccolo contributo in questa direzione.

Capitolo 2

Tecnologie per la gestione dei Big Data e l'analisi testuale

I Software per la manipolazione di Big Data a disposizione degli analisti sono molteplici e sovente occorre selezionare con accuratezza quelli che si prestano allo scopo da raggiungere. Il nostro obiettivo, come anticipato, è quello di realizzare un'analisi dei contenuti con supporti *Open Source* che consentano un'esecuzione *distribuita* del codice e che permettano un abbattimento notevole dei tempi di realizzazione.

2.1 La scelta del database.

Innanzitutto occorre provvedere allo storage dei dati acquisiti, ovvero occorre strutturare un database. Per i nostri scopi, si è deciso di adottare un database NoSQL, nella fattispecie *MongoDB* ([MonDB]).

Per comprendere i vantaggi di questa scelta nel caso specifico da noi esaminato, dobbiamo confrontare le caratteristiche di questi database con quelle dei database dai quali, già a partire dallo stesso nome, così nettamente si diversificano e cioè i database relazionali gestiti con linguaggio SQL [1]. Questi ultimi sono spesso indicati con la sigla RDBMS [2].

Di seguito analizziamo le caratteristiche che giustificano la nostra scelta di utilizzo.

2.1.1 Scalabilità.

La differenza principale sta nel concetto di *scalabilità*. Per scalabilità di un database si intende la capacità di gestire la crescente mole di informazione registrata.

[1] Che sta per *Structured Query Language*
[2] Che sta per *Relational DataBase Management System*

I database possono avere scalabilità orizzontale o verticale. Con quest'ultimo termine si intende che il loading di ulteriori dati può essere gestito aumentando le prestazioni degli hardware (come RAM, CPU, SSD, ecc.); in generale, dunque, la scalabilità verticale si traduce nella necessità di aumentare le risorse. Questa caratteristica è tipica dei database SQL.

Viceversa per i database a scalabilità orizzontale è possibile collegare, ad esempio, più server per gestire la memorizzazione dei dati. Un'analogia utile è quella di considerare questi supporti come nodi di un grafo: la scalabilità orizzontale consente di gestire il flusso di dati semplicemente aggiungendo nodi, senza dunque richiedere prestazioni superiori alle macchine. Questa gestione è adottata dai database NoSQL ed è appena il caso di notare che ciò è coerente con la nostra impostazione di tipo distribuito.

2.1.2 Struttura di memorizzazione.

Nello specifico un database relazionale registra i dati in forma tabulare mentre esistono database NoSQL con diverse architetture di registrazione. Nel caso di MongoDB, si tratta di un database *document-oriented*, ossia la registrazione dei dati avviene accorpando tutte le informazioni in documenti, ognuno dei quali risulta perciò un'unità indipendente dalle altre.

Inoltre un documento, rispetto alla tabella, non possiede una struttura fissata (*Unstructured Data*) e questo si traduce in una veloce memorizzazione, in quanto non occorre conoscere in anticipo la struttura dell'input, e in un minore costo computazionale per eventuali trasferimenti.

Per di più i dati provenienti da Internet (in particolare dai Social Network) non hanno sempre una struttura fissata.

In generale e per completezza la mancanza di struttura non consente la gestione di query complesse, tuttavia il sistema di indicizzazione è efficiente in entrambi i casi e dunque rimane comunque possibile richiamare i documenti necessari in modo rapido.

2.2 Sistemi di gestione distribuiti e Apache Spark.

La scelta di utilizzare Apache Spark (di seguito solo Spark) come shell da cui lanciare gli script per l'analisi, si fonda soprattutto sulla capacità di gestire diversi compiti in parallelo. In generale, sebbene su modeste quantità di dati non si ottiene un guadagno sensibile rispetto ad altri programmi in termini di tempo di esecuzione, per la gestione di big data con Spark si ottengono risultati ragguardevoli rispetto ad altri software ([QuoSfH]).

Operare con linguaggio Python (scelta che motiveremo in seguito), rallenta notevolmente ([QuoSvP]) l'esecuzione rispetto all'utilizzo con il linguaggio nativo Scala, che meglio si adatta ad un ambiente Java, ma, ciononostante, rispetto ad altri software i tempi di esecuzione sono comunque di molto inferiori.

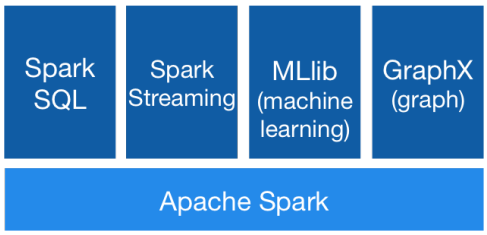

Figura 2.1: Componenti di Spark

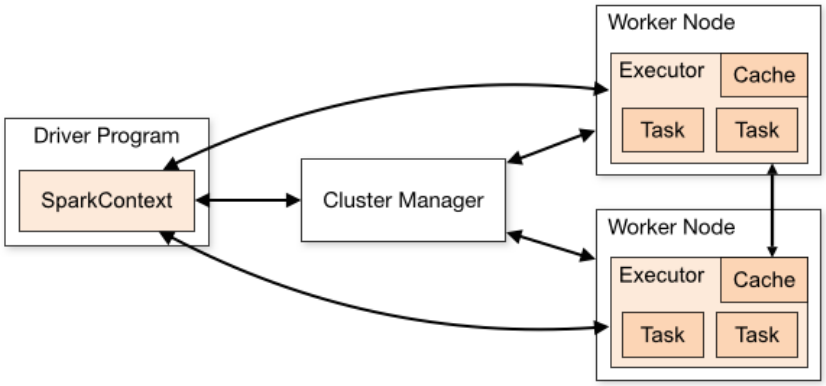

Figura 2.2: Gestione dell'esecuzione in Spark

Poiché un aspetto importante dell'analisi è rappresentato dall'ambiente informatico in cui essa si sviluppa, parleremo più diffusamente di Spark, descrivendo le principali caratteristiche di interesse che ritroveremo sia direttamente che indirettamente nello svolgimento.

2.2.1 Struttura e funzionamento di Spark

Apache Spark è stato sviluppato all'Università di Berkeley ed è stato rilasciato nel 2014.

Essenzialmente Spark è un sistema di computazione cluster distribuito e ad alte prestazioni. In figura 2.1 possiamo vederne i principali costituenti.

Di particolare interesse sono*Spark SQL*, che rappresenta il modulo che consente di lavorare con dati strutturati, e *MLlib* che sta per *Machine Learning Library* ed è una libreria che contiene diversi algoritmi per il machine learning. In quest'ultima troviamo anche alcuni algoritmi di topic modeling, tra cui quello che esegue l'LDA e di cui parleremo diffusamente più avanti.

Tutte queste aree sono sottese, in figura, da *Apache Spark*, il cui core controlla le esecuzioni di ciascuna componente. Vediamo nel dettaglio il modo in cui Spark esegue le istruzioni nella figura 2.2 .

In essa notiamo che il *Cluster Manager* centrale gestisce il *Driver Program*, richiamato mediante lo *Spark Context*, e i diversi *Worker Nodes*. In particolare ciascun *Worker Node* esegue diverse *Tasks*, cioè compiti, assegnati. Se un nodo va in crash, esso viene ripristinato dal *Driver*. In questa gestione si ravvisa lo sviluppo parallelo dell'esecuzione con Spark. Tra i diversi compiti eseguiti c'è l'avvio dei e la gestione dei Database Resilienti. Di essi parleremo diffusamente nei prossimi paragrafi poiché rivestono un ruolo importante nel prosieguo.

2.2.2 Resilient Distributed Dataset

La struttura di base in cui i dati vengono incapsulati in Spark è quella dei *Resilient Distributed Dataset* (RDD), la cui caratteristica principale è quella di rimanere immutati durante tutto il processo di esecuzione. Questo vuol dire che anche in caso di una operazione su di essi, il programma non modifica il dataset resiliente, bensì costruisce un altro RDD. Questo rende l'intero processo estremamente robusto e preserva i dati da eventuali crash o modifiche indesiderate, in quanto ogni volta l'insieme dei dati è praticamente ricostruibili.

Esistono due modi di creare un RDD: riferendosi ad un sistema di memorizzazione esterno, oppure usando il comando *parallelize* su una collezione già presente sul Driver. Quest'ultimo comando copia, sostanzialmente, i dati che processa e permette la loro esecuzione in parallelo.

Le operazioni che è possibile eseguire sui RDD sono di due tipi: le *trasformazioni*, che creano un nuuovo dataset a partire da uno preesistente, e le *azioni*, che restituiscono un valore al Driver dopo aver operato un calcolo sul dataset. Un esempio di trasformazione è *map* che processa secondo una funzione ogni elemento di un dataset e restituisce un nuovo RDD costituito dai risultati; un esempio di azione è, invece, il comando *reduce*, che aggrega tutti gli elementi del RDD e restituisce un risultato finale al Driver.

Tutte le trasformazioni in Spark non vengono eseguite fino a quando non si necessita di un risultato effettivo nel Driver (ad esempio da parte di un'azione). In questo modo (definito *lazy*, cioè pigro) si ottimizzano i tempi di esecuzione.

2.2.3 Direct Acyclic Graphs

Tutte le trasformazioni vengono eseguite come *Direct Acyclic Graphs*, ovvero come grafi aciclici. Essa è un'astrazione che consente di visualizzare propriamente il processo. In figura 2.3 possiamo vedere con un DAG appare, si comprende la struttura aciclica che ricorda quella ad albero, e si nota l'attitudine al lavoro in parallelo. Osserviamo come nei diversi passaggi (o *stage*) si eseguano diverse operazioni.

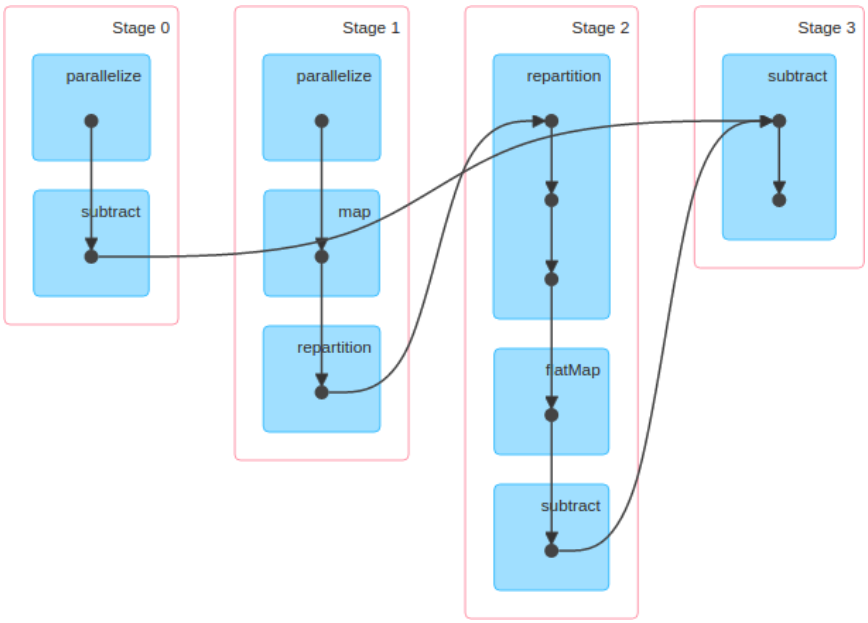

Figura 2.3: Esempio di visualizzazione di un DAG

È appena il caso di ricordare che, ogni volta che è necessario, il DAG si può ricostruire a partire dal Driver, conserva, cioè, la struttura resiliente.

2.3 Il linguaggio Python e le sue librerie

Il linguaggio Python è certamente tra i più usati nel campo dell'analisi dei dati. Il suo maggiore punto di forza consiste nella enorme quantità di librerie che consentono di eseguire con pochi comandi pressoché qualsiasi procedura analitica e statistica; inoltre consente di interfacciarsi a diversi tipi di software. Proprio per questo, nella nostra analisi, il linguaggio Python rappresenta il nerbo di collegamento tra i diversi moduli di cui è costituita la procedura realizzata. Approfondiamo caso per caso le librerie prese in considerazione.

2.3.1 Python e web scraping con Scrapy

I dati esaminati provengono dalla rete, per questo è necessario implementare una procedura che permetta la rilevazione e la memorizzazione di questi ultimi in modo automatizzato. L'attività di rilevazione sistematica di informazione da specifiche fonti Internet viene definita *web scraping* e un programma che effettui lo scraping viene definito *scraper* o *spider*. Nel nostro caso, si è fatto ricorso a *Scrapy* [Scrpy], che è costituito da diversi moduli che consentono di estrarre contenuti da pagine web ovvero di costruire uno o più

spider. Esso consente di effettuare il parsing del linguaggio HTML/XML (o in alternativa usare i selettori CSS e le espressioni XPath) con cui è costruita la pagina internet e registrare le informazioni desiderate. La praticità dell'utilizzo di una piattaforma con funzionalità built-in com'è Scrapy, piuttosto che l'uso di una singola libreria di parsing, com'è l'eccellente *BeautifulSoup* ([BeaSp]), sta nella naturalezza con cui si svolgono alcune operazioni, tra le quali, fondamentale per la nostra analisi, quella di connettersi a MongoDB mediante un pacchetto che vediamo in dettaglio.

2.3.2 PyMongo

La memorizzazione dei dati ottenuti è il secondo passo da compiere. Abbiamo già esaminato in dettaglio il Database NoSQL MongoDB; per connettersi ad esso con Python si usa la libreria *PyMongo* (si veda [PyMDB]). Mediante le funzionalità che essa consente è possibile non solo registrare in una specifica collection in dati ottenuti mediante lo scraping delle fonti, ma anche e soprattutto evitare i duplicati. Nell'ipotesi di una raccolta informazioni sistematica, poniamo giornaliera, è, infatti, ragionevole supporre che su una stessa pagina web siano presenti dati risalenti al giorno prima e che dunque si possiedono già in memoria. Gestire i duplicati è dunque una fase importante: in generale, questo si può fare verificando se il nuovo dato possiede qualche caratteristica identica a un dato già memorizzato. Ad esempio, nel caso della raccolta di articoli di giornale che opereremo, possiamo controllare se un articolo possiede un titolo identico ad uno già memorizzato; potremmo verificare, per una maggiore sicurezza, che abbia anche il testo identico, tuttavia è estremamente raro che due diversi articoli di giornale possiedano uno stesso titolo. Come si vedrà in seguito, abbiamo adottato la prima linea di condotta, ovvero l'esclusiva verifica del titolo e non solo per le ragioni appena esposte: una verifica anche del testo operata su larghi dataset ha un costo computazionale decisamente non trascurabile.

2.3.3 *Natural Language Toolkit*

Il *Natural Language ToolKit* (di seguito *nltk*) è un insieme di librerie (si veda [NaLTK]) che consente di processare il linguaggio naturale.

La sua importanza nell'analisi è giustificata dalla considerazione che ogni file di testo porta con sé una quantità enorme di *materiale-spazzatura* che rallenta, e in qualche caso distorce, l'analisi che si sta eseguendo.

Pertanto occorre innanzitutto rendere omogeneo l'intero documento, trasformando tutto il testo in formato minuscolo ed eliminando segni di punteggiatura, apostrofi ma anche numeri e caratteri speciali, spesso presenti come residui del linguaggio informatico di impaginazione o come simboli matematici e di interpunzione.

Ciò fatto, occorre eliminare le *stopwords*, ovvero articoli, preposizioni semplici o articolate e congiunzioni, quelle parole, cioè, che sono poco significative ma che compaiono sovente all'interno di una frase.

Ad esempio, i versi:

```
Sempre caro mi fu quest'ermo colle,
E questa siepe, che da tanta parte
Dell'ultimo orizzonte il guardo esclude.
```

dopo un'operazione di omogenizzazione, pulitura e stopwords removing, diventerebbe

```
caro fu ermo
siepe tanta parte
ultimo orizzonte guardo esclude
```

Per effettuare un'operazione di stopwords removing occorre avere a disposizione dizionari ben forniti, che, nel caso specifico della lingua italiana e a differenza di quella anglosassone, non abbondano. La lista di termini di default in *nltk* è un buon compromesso anche perché è consentito aggiungervi elementi qualora manchino.

In generale sembrerebbe che una tale procedura sia perfettamente legittima nell'ottica dell'analisi testuale e non comporti nessun costo in termini di perdita di informazione: la prima affermazione è certamente vera, sulla seconda bisogna essere cauti. Nel caso esaminato la parola *colle*, nel senso di collina, rilievo, viene eliminata dal primo verso perché confuso con la preposizione articolata *con + le*. Sebbene in questo caso la perdita di informazione sia minima, può succedere di peggio: supponiamo, ad esempio, di analizzare un documento che tratta di Intelligenza Artificiale e in cui sovente troviamo la sigla *AI* ad indicarla lungo il discorso. Se riduco tutte le maiuscole a minuscole, la sigla diviene *ai* la quale, applicando lo stopwords removing, viene completamente eliminata dal documento in quanto confusa con una preposizione articolata. Pertanto uno dei termini-chiave del nostro documento viene inevitabilmente perso!

In generale, i casi in cui le procedure di stopwords removing comportano una gravosa perdita di informazione sono rari e, alla lunga e su un gran numero di dati, tali procedure comportano più benefici che svantaggi; tuttavia occorre sempre tenere in conto che non è mai possibile ridurre il contenuto di un documento senza che ciò comporti anche una certa perdita di informazione.

La fase successiva consiste nell'applicare un *word stemmer*, il cui ruolo è ridurre una parola alla sua radice semantica; ad esempio le varie coniugazioni di un verbo alla radice ('vado', 'sarò andato' → 'andare'), oppure parole con una stessa etimologia o affinità di significato ('giocatore', 'giocattolo'→ 'gioco').

Per un'analisi efficiente, in questo caso, occorrerebbero dei dizionari molto completi che, a differenza del caso precedente, per lo stemming mancano del tutto.

La soluzione, implementata in nltk, consiste in uno stemming forzato, ovvero in una riduzione sommaria dei vocaboli a radici *fittizie* ('andiamo'–> 'and') che pertanto non permette né di rilevare sempre la corretta origine semantica di un vocabolo, né di distinguere due vocaboli diversi nel significato ma con uguale 'radice' ('computazionale', 'computer'–> 'comput'). Lo stemmer riconosce, dunque, soltanto i suffissi più comuni e li tronca dal resto della parola.

Sebbene in questo caso la perdita di informazione sia necessariamente più marcata, questa procedura è necessaria e incrementa, generalmente, la conoscenza acquisita nel risultato finale.

Da un punto di vista delle analisi descritte nel Capitolo 1, riconosciamo che siamo sicuramente in grado di eseguire un'analisi lessicale, e parzialmente quella sintattica e semantica. L'analisi sintattica viene operata in modo tacito, ovvero escludendo quelle *stopwords* appartenenti ad una specifica categoria di parole (come articoli e congiunzioni). L'analisi semantica, invece, è quella che ha più margine di miglioramento, in quanto la sola operazione di *stemming*, come abbiamo visto non è ottimale. Entrambe queste analisi sono ostacolate dall'ambiguità intrinseche di alcuni aspetti della lingua, per cui occorrerebbe esaminare ciascun termine specifico nel suo contesto originario per poter comprendere il ruolo che esso svolge nella frase è il suo effettivo significato. Il fatto che queste tipologie di analisi non vengano approfondite al massimo delle loro potenzialità non è dovuto solo alla mancanza di strumenti (come i dizionari) o alla constatazione che, comunque, qualsiasi metodo di comprensione non sarà che approssimato: esso è contingente nell'ottica di un approccio *bag-of-words* delle successive analisi, in cui, cioè, il contesto proprio di ciascuna parola viene trascurato.

2.3.4 Scikit-Learn e pyspark

Il cuore dell'analisi è costituito dalle procedure di *topic extraction* che si applicano ai dati memorizzati. Gli algoritmi verranno richiamati attraverso due librerie principali: *Scikit-Learn* e *pyspark* (si veda rispettivamente [SckLn] e [PySpk]).

La prima libreria consente di richiamare diverse procedure per il machine learning ed è diffusamente usata nel campo dell'analisi dei dati. Da essa richiamiamo principalmente i seguenti algoritmi:

1. TfidfVectorizer: ovvero la procedura dalla quale si ottiene una matrice DTM mediante lo schema *tdidf*;

2. TruncatedSVD: che consente la decomposizione in valori singolari di una matrice (SVD sta per Singular Value Decomposition).

Nel prossimo capitolo comprenderemo come essi si incasellano nell'intera procedura.

Per quanto riguarda *pyspark*, esso rappresenta il collegamento con la shell di Spark. Mediante *pyspark* possiamo richiamare lo *SparkContext* ovvero il collegamento al cluster di Spark. Mediante esso siamo in grado di creare un RDD e processarlo, ed eseguire il codice come se ci trovassimo in ambiente Spark. Per richiamarlo, basta digitare le seguenti linee di codice:

```
import pyspark
from pyspark import SparkContext
sc=SparkContext()
```

Solamente uno *SparkContext* alla volta può essere avviato e di questo si terrà conto durante lo sviluppo della procedura. Questo perché solamente uno *SparkContext* può essere attivo sulla *Java Virtual Machine* (JVM), che rappresenta l'ambiente in cui il codice viene eseguito. Per stopparlo, possiamo digitare semplicemente

```
sc.stop()
```

Mediante lo *SparkContext* saremo in grado di richiamare da *MLlib* il programma per l'esecuzione del topic model *Latent Dirichlet Allocation*; di esso e dei topic model in generale ci occuperemo nel seguente capitolo.

Capitolo 3

Modelli di Topic Extraction. LSI e LDA.

Il fulcro dell'analisi che andremo a svolgere è costituito dalla capacità di estrarre dai testi analizzati informazioni concernenti il loro contenuto: tale è il compito dei *topic model* Ci concentreremo principalmente su due estrattori, il *Latent Semantic Indexing* ed il *Latent Dirichlet Allocation*, di seguito abbreviati LSI e LDA.

Storicamente, l'LSI rappresenta il capostipite dei *topic model*: tra la fine degli anni '80 e l'inizio dei '90, nell'ambito del cosiddetto *Information Retrieval* (Recupero delle Informazioni), viene definito il *Latent Semantic Indexing* da Deerwester et al. [Der90]. Tale metodologia è stata adottata fino alla fine del secolo scorso nell'ambito dei motori di ricerca. Successivamente, Hofmann [Hof99] nel 1999, ha generalizzato la procedura implementando il *Probabilistic Latent Semantic Indexing* (pLSI) di cui daremo qualche cenno. Infine, nel 2003, Blei et al. [Ble03] hanno realizzato l'algoritmo di LDA, che ha raggiunto grande fama e diffusione.

Definiamo gli strumenti principali che utilizzeremo in entrambe le analisi.

Sia v un vettore, in seguito v^1 indicherà la prima componente, v^2 la seconda, v^i la i-esima e così via.

1. Un dizionario è un insieme finito del tipo $\{1, \dots, V\}$, dove V ne rappresenta anche la lunghezza;

2. Una *parola* è un'unità del dato discreto. La parola i-esima si indica con un vettore binario di lunghezza V, ovvero, indicando la parola con w, l' indice i è tale per cui $w^i = 1$ mentre per ogni $j \neq i$ $w^j = 0$.

   Due differenti parole hanno, chiaramente, diversi vettori associati.

3. Un *documento* è una sequenza di N parole e lo si denota con $\mathbf{w} = (w_1, w_2, w_N)$, dove con w_n si intende l'n-esima parola della sequenza.

4. Un *corpus* è un insieme di M documenti e lo indichiamo con la scrittura D $=\{\mathbf{w_1}, \mathbf{w_2}, \ldots, \mathbf{w_M}\}$.

3.1 Latent Semantic Indexing.

Il *Latent Semantic Indexing*, spesso chiamato anche *Latent Semantic Analysis*, è una metodologia di tipo *bag of words* che consiste nella riduzione dimensionale di una matrice parole-documenti.

3.1.1 Schema *tfidf*.

Definiamo la seguente matrice

$$DTM^1 = \{u_{ij}\}_{i=1,2,\cdots,V;j=1,2,\cdots,M}$$

ove V rappresenta il numero totale dei termini presenti in tutti i documenti e M il numero di tali documenti. Pertanto l'elemento u_{ij} rappresenta il peso che riveste il termine i nel documento j. Come sistema di pesi, si potrebbe adottare la semplice frequenza di un termine nel documento, tuttavia si ricorre nella maggior parte dei casi, all'utilizzo della funzione *tfidf*[2], definita per la prima volta in [Sal88], che ha la seguente forma:

$$u_{ij} = tfidf(i,j) = n_{ij} \cdot \log\left(\frac{M}{n_j}\right)$$

anche se sovente si preferisce un'analoga formula normalizzata di tipo coseno (che ricorda il prodotto scalare)

$$u_{ij} = \frac{tfidf(i,j)}{\sqrt{\sum_{t=1}^{T} tfidf^2(i,j)}}.$$

In ogni caso, n_{ij} rappresenta la frequenza relativa del termine i-esimo nel documento j-esimo e n_j è il numero di documenti che contengono il termine i-esimo; Ciò vuol dire che il peso (o la *rilevanza*) di un termine per un documento è direttamente proporzionale alla sua frequenza in quel documento e inversamente proporzionale alla sua frequenza nell'intera collezione di documenti.

In generale lo schema tfidf pesa maggiormente i termini più rilevanti nei documenti.

[1] Sta per *Document Term Matrix*
[2] Che sta per term frequency-inverse document frequency

3.1.2 Descrizione del modello LSI

A questo punto si opera la riduzione dimensionale della matrice DTM in modo da identificare un sottospazio lineare in grado di spiegare il più possibile dell'intera varianza del corpus di documenti. Si dimostra,infatti, che è sempre possibile scrivere la matrice DTM come:

$$DTM = U\Sigma V^t$$

dove U, e V sono matrici ortogonali e Σ è la matrice diagonale dei valori singolari di DTM:

$$\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_r \end{bmatrix}$$

con r rango di DTM e $\sigma_1 \geq \sigma_2 \geq \dots, \sigma_r$. Sostituendo alla matrice Σ, un'altra matrice diagonale Σ^* che contiene soltanto i primi k valori singolari più elevati, con $k \leq r$, e annullando i rimanenti, si effettua un'approssimazione di DTM che consiste in una proiezione ortogonale di DTM su un sottospazio lineare di dimensione inferiore.

In formule

$$DTM^* = U\Sigma^* V^t \sim U\Sigma V^t = DTM$$

L'approssimazione DTM* ha rango k.

Con l'LSI si riescono a cogliere anche alcuni aspetti importanti del linguaggio naturale, come la sinonimia e la polisemia e in generale costrutti semantici latenti (da cui il nome della procedura), oltre che notevoli compressioni dei documenti originali, tuttavia l'LSI non modellizza la genesi dei dati.

3.1.3 Un esempio di applicazione

Per testare l'efficacia dell'algoritmo LSI, strutturiamo un esempio di applicazione su un corpus costituito da due documenti. Nel caso in questione abbiamo selezionato le biografie di Isaac Newton ed Albert Einstein che sono pubblicamente fruibili su Wikipedia (vedi [BioEN]). Essi vengono ripuliti e poi sottoposti ad un operazione di rimozione delle stopwords: l'estensione dei file è di circa 2000 parole per entrambi. Il codice che processa secondo l'LSI è il seguente

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD

vectorizer = TfidfVectorizer(use_idf=True, ngram_range=(1, 3))
X = vectorizer.fit_transform(collection_testi)
```

```
lsa = TruncatedSVD(n_components = 1, n_iter = 100)
lsa.fit(X)
```

dove *collection_ testi* è una lista che contiene le biografie processate. Dal pacchetto *sklearn* abbiamo importato sia l'algoritmo *tfidf*, sia l'algoritmo *TruncatedSVD* che prima fattorizza la matrice parole-documenti e poi ne riduce il rango. Quest'ultimo supporta due tipi di algoritmi: un solver di SVD rapido e randomizzato e un algoritmo naive che usa ARPACK come solutore.

Come si vede, abbiamo impostato a 1 il numero di topic: in tal caso ci aspettiamo che l'output evidenzi la comunanza tra i due file di testo, ovvero ciò che in comune hanno Isaac Newton e Albert Einstein. L'output è il seguente:

```
Topic 1:
einstein
newton
studi
anni
teoria
fisica
stato
anno
relatività
finire
```

in linea con le nostre aspettative. Abbiamo omesso le probabilità associate a ciascun termine: è importante solamente constatare che le parole vengono elencate in ordine decrescente di rilevanza nei confronti del topic.

Nel caso in cui impostiamo a 2 il numero di topic da restituire, ci aspettiamo che ci venga restituito un output in cui si differenzi un gruppo di parole riguardanti la vita di Einstein e un gruppo relativo a quella di Newton, che rappresentano i due argomenti principali di cui si sta parlando. In questo caso l'output (parziale) è

```
Topic 1:
einstein
newton
studio
teoria
fisico
...

Topic 2:
newton
```

```
mela
divenne
isaac
\textbf{royal society}
zecca
...
```

Osserviamo un fatto interessante: sebbene le parole vengano memorizzate singolarmente, il programma riconosce che *royal* e *society* costituiscono un'unica locuzione. Questo perché le occorrenze di entrambe le parole sono le stesse e il programma rileva che hanno la medesima comunanza nel testo. In generale possiamo anche non richiedere all'algoritmo di rilevare questi costrutti, basta lasciare l'opzione *ngram_range* di *TfidfVectorizer* con le impostazioni di default.

Poiché l'LSI è un algoritmo deterministico, l'intera procedura e l'output sono perfettamente riproducibili e si possono confrontare i risultati con quelli appena mostrati. I file si trovano nella repository Github descritta nell'introduzione.

Altri esempi di applicazione dell'algoritmo LSI si possono trovare nell'articolo di Deerwester [Der90] e nel riferimento [San15], dove si applica l'LSI all'intero corpo di articoli di Wikipedia.

3.1.4 Cenni al *pLSI*

Pur senza addentrarci nei dettagli, diamo qualche cenno del pLSI, in quanto esso rappresenta un passo intermedio tra l'LSI e l'LDA.

Come abbiamo visto, il modello LSI consente di cogliere un nucleo conoscitivo di un documento o di una collezione di documenti ma non è un modello generativo, nel senso che non coglie l'aspetto statistico della distribuzione delle parole nel documento.

Il modello pLSI, anch'esso basato sull'approccio *bag-of-words*, consente di risolvere questo problema aggiungendo, di fatto, un modello generativo all'LSI: si vuole assegnare una distribuzione di probabilità congiunta alla coppia documento-parola, in particolare si suppone che ogni parola sia generata indipendentemente dalle altre da un modello di mistura. Per far ciò, si considera un'ulteriore variabile, poniamo z (che rappresenta la classe latente), condizionatamente alla quale la probabilità congiunta si decompone nel prodotto della distribuzione sui documenti e sulle parole.

Con queste assunzioni, ne risulta un modello generativo dei parametri (documento, z e parola); l'inferenza sui parametri si ottiene massimizzando la funzione di log-verosimiglianza mediante l'algoritmo EM.

Sebbene il pLSI sia un primo passo importante nella direzione dei modelli probabilistici, esso possiede ancora un ampio margine di miglioramento, considerato che il modello probabilistico non arriva al livello dei documenti.

In sostanza il pLSI associa ad ogni documento una lista di valori, che sono le proporzioni di mistura per i topic e dunque non c'è nessun modello generativo per questi valori. Da un lato questo implica una crescita delle proporzioni di mistura che è lineare rispetto alla dimensione del corpus, dall'altro lascia nell'incertezza sull'assegnazione delle probabilità a un documento esterno al dataset di training.

Tali problemi sono superati dal LDA come vedremo in seguito.

3.2 Latent Dirichlet Allocation

Il modello LDA è di tipo Bayesiano, gerarchico su tre livelli, adatto in particolare al processing di file di testo, ma più in generale in grado di processare dati discreti. Così come l'LSI si tratta anch'esso di un algoritmo di tipo *bag-of-words*, ma al contrario di quest'ultimo, non si concentra sull'aspetto di riduzione dimensionale di una matrice di parole-documenti, bensì cerca, in qualche senso che sarà più chiaro in seguito, di ricostruire la struttura di un documento sulla base di topic supposti latenti, in numero fissato e con una distribuzione predefinita.

3.2.1 Formalizzazione del modello.

Per ogni documento $\mathbf{w}$ in un corpo D valgono le seguenti assunzioni:

1. Sia $\theta \sim Dir(\alpha)$
2. Per ognuna delle N parole w_n:
   - Si scelga un topic $z_n \sim Mult(\theta)$;
   - Si scelga una parola w_n da $p(w_n|z_n, \beta)$, ovvero una distribuzione Multinomiale condizionata a z_n

   In generale assegnare una particolare distribuzione al numero di parole N non è vincolante per il prosieguo, tuttavia per fissare le idee poniamo
3. $N \sim Poiss(\eta)$

In sostanza, riassumendo, abbiamo un modello che concepisce i documenti come misture di topic che sono latenti e ciascun topic, a sua volta, è inteso come una distribuzione Multinomiale sui termini del dizionario V. Tale modellizzazione agisce su tre livelli, come avevamo specificato in precedenza, che analizziamo più nel dettaglio.

Il primo livello è rappresentato dalle parole nel documento, a ciascuna delle quali è associata una classe latente $z_n \in Z = \{1, 2, \cdots, K\}$, chiamata anche *topic assignment* nella relazione 3b:

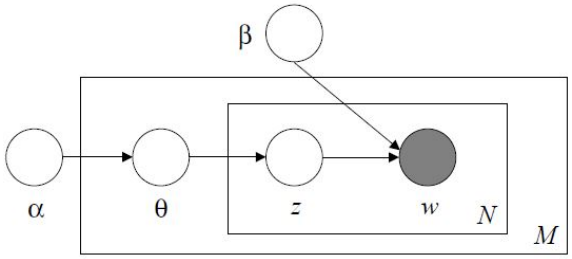

Figura 3.1: Struttura generativa dell'LDA

$$w_n \sim p(w_n|z_n, \beta) \equiv Mult(\beta_{zn})$$

In cui β è una matrice di topic tale per cui $\beta_{ij} = p(w^j = 1|z^i = 1)$ e che ha dimensione $K \times V$. Nella relazione dunque, la variabile multinomiale ha come parametro il vettore riga corrispondente a z_n. La matrice β nel nostro modello rappresenta un parametro da stimare, inoltre osserviamo che il numero di topic K è fissato.

Il secondo livello è rappresentato dal documento stesso, la cui relazione specificata è quella al punto 3a. Il parametro θ viene indicato anche con il nome di *topic proportions*.

Il terzo e ultimo livello consiste nel corpus dei documenti. Per ogni documento $\mathbf{w} \in D$ abbiamo visto che θ viene generato secondo una distribuzione di Dirichlet di parametro α la cui forma esplicita è la seguente:

$$p(\theta|\alpha) = \frac{\Gamma(\sum_{k=1}^{K} \alpha_k)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \theta_1^{a_1-1} \theta_2^{a_2-1} \cdots \theta_K^{a_K-1}.$$

La scelta della distribuzione di Dirichlet è giustificata dal fatto che appartiene alla famiglia esponenziale, ha un buon comportamento sui simplessi di dimensione $K-1$ (abbiamo $\theta \geq 0$ e $\sum \theta = 1$), ha statistiche sufficienti di dimensione finita ed è coniugata alla distribuzione Multinomiale; pertanto risulta una scelta efficace dal punto di vista dell'implementazione sia teorica che pratica. Notiamo infine, che in questo modo proporzioni dei topic all'interno di ogni documento sono generate indipendentemente dal documento stesso e questo comporta sì la stima di un parametro aggiuntivo, ma consente di liberarsi dall'etichettatura dei documenti.

Nella figura 3.1 è riassunto il processo generativo appena esposto.

La distribuzione congiunta della topic proportion θ, dell'insieme di N topic $\mathbf{z}$ e dell'insieme di N parole w, dati i parametri α e β è la seguente:

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

dove $p(z_n|\theta)$ è semplicemente θ_i per quell'unico indice per cui anche $z_n^i = 1$. Integrando rispetto a θ abbiamo la marginale

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

Infine, otteniamo la probabilità del corpus è data dalla produttoria delle marginali:

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

Quest'ultima relazione è il punto di partenza da cui si costruisce la funzione di verosimiglianza da massimizzare per fare inferenza sui parametri di interesse.

3.2.2 Inferenza sui parametri.

Il nocciolo del problema è rappresentato dal fatto che la massimizzazione della log-verosimiglianza

$$\sum_{v=1}^{V} logp(w_v|\alpha,\beta),$$

è sostanzialmente intrattabile dal punto di vista computazionale in quanto, scrivendo diversamente la marginale, abbiamo che

$$p(\mathbf{w}|\alpha,\beta) = \frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)}\int\left(\prod_{i=1}^{k}\theta_i^{\alpha_i-1}\right)\left(\prod_{n=1}^{N}\sum_{i=1}^{k}\prod_{j=1}^{V}(\theta_i\beta_{ij}^{w_n^i})\right)d\theta$$

e si dimostra che la presenza del prodotto delle variabili θ e β non consente una soluzione.

Per ovviare a questa impossibilità, data la convessità delle funzioni in gioco, si ricerca un limite inferiore della funzione di log-verosimiglianza e si ricorre all'algoritmo VEM (Variational Expectation Maximization); il lower bound è costituito da un'approssimazione della distribuzione a posteriori sulle variabili latenti che appartiene alla seguente famiglia

$$q(\theta, z|\gamma,\varphi) = q(\theta|\gamma)\prod_{n=1}^{N} q(z_n|\varphi_n)$$

A questo punto l'algoritmo VEM consta di due passaggi

1. E-step: Per ogni documento d in D si trovano i valori ottimali dei parametri $\gamma*$ e $\phi*$ minimizzando la divergenza di Kullback -Leibler

tra la distribuzione variazionale e la distribuzione a posteriori vera, in simboli

$$(\gamma^*, \varphi^*) = \min_{\gamma,\varphi} D(q(\theta, z|\gamma, \varphi)||p(\theta, \mathbf{z}|\mathbf{w}, \alpha, \beta))$$

2. M-step: si massimizza rispetto ad α e β il lower-bound trovato al passo precedente.

Questi due passi sono ripetuti fino a quando l'algoritmo converge ad un massimo locale della funzione di log-verosimiglianza.

L'algoritmo LDA richiamato in Spark, ricalca le linee teoriche appena enunciate: esso possiede un ottimizzatore (*EMLDAOptimizer*) che pratica l'algoritmo EM sulla verosimiglianza.

Esistono anche altri algoritmi per effettuare inferenza sui parametri: uno di quelli che ha ricevuto più attenzione è il *Gibbs Sampling* che sfrutta un approccio differente ed è un tipo di algoritmo Monte Carlo Markov Chain. Non approfondiremo i dettagli di questa procedura, che abbiamo citato per completezza. Si rimanda a [Dar11].

3.2.3 Esempio di applicazione

Per verificare come opera l'algoritmo di LDA, utilizziamo gli stessi testi a contenuto bibliografico adoperati anche per l'LSI. Nel caso di un topic per ciascun articolo abbiamo:

```
Articolo su 'Einstein':
einstein
studi
teoria
...
stato
relatività
milano

Articolo su 'Newton':
newton
anni
mela
...
legge
egli
volt
```

Anche i questo caso ciascuna parola è affiancata da una probabilità che misura affinità con il topic in questione. Le parole sono posizionate in ordine decrescente rispetto a questa probabilità. Si osserva una notevole somiglianza

con l'output dell'LSI, probabilmente dovuta alla breve estensione dei file di testo analizzati e alla specificità del testo analizzato.

3.2.4 Confronti e considerazioni

La differenza principale tra i due modelli presi in esame consiste nel fatto che mentre l'LSI è un algoritmo deterministico che non modellizza alcuna struttura generativa, l'LDA è un algoritmo probabilistico in grado di modellizzare tale struttura. I risultati di quest'ultimo possono variare ogni volta che si esegue l'algoritmo proprio in virtù dell'impostazione pseudo-casuale.

Risulta altresì chiaro che il modello LSI è molto semplificato e meno generale dell'LDA: come in ogni scenario, però, occorre valutare sapientemente lo strumento che conviene utilizzare in una determinata situazione. Non sempre lo strumento più generale si rivela la scelta migliore; può capitare, ad esempio, che risulti troppo oneroso in termini di complessità computazionale rispetto ad un modello più semplificato, pur restituendo un output non dissimile da quest'ultimo (lo abbiamo visto nell'esempio di applicazione per entrambi). Nel prossimo capitolo daremo una giustificazione della procedura adottata nel nostro algoritmo, che fonde LDA e LSI sulla base delle loro caratteristiche.

L'LSI e l'LDA possono entrambe essere utilizzate per procedure di apprendimento non supervisionato.

Un'ultima riflessione sui metodi di tipo *bag-of-words*. In [Ble03] troviamo un'importante considerazione a proposito della facoltà di trascurare l'ordine delle parole (e nel caso dei corpus anche quello dei documenti). Il ragionamento che ha condotto alla ideazione dell'LDA parte dal teorema di De Finetti [DeFin] che afferma la distribuzione congiunta di una sequenza di variabili aleatorie infinitamente scambiabili possiede una rappresentazione in termini di mistura, condizionatamente ad un parametro latente: per questo se si vuole avere una rappresentazione della scambiabilità sia per i documenti che per le parole, dobbiamo considerare una mistura che va bene per entrambi. Ribadiamo che la scambiabilità non implica indipendenza e identica distribuzione, quanto piuttosto indipendenza e identica distribuzione *condizionatamente* ad un parametro latente. Per questo, se condizionata, la distribuzione congiunta delle variabili aleatorie si può notevolmente semplificare.

Nel caso dell'LDA questo consente di elevarsi ad un livello più alto rispetto al pLSI, ovvero di estendere anche a corpus di documenti il modello e la procedura.

Capitolo 4

Progettazione e sviluppo dell'algoritmo

Una volta elencati gli strumenti occorre assemblarli in una procedura organica, composta di più moduli, la quale soddisferà i seguenti requisiti:

1. Raccoglie le informazioni dalla rete e li memorizza in un database;

2. Ripulisce i file di testo e li prepara per l'operazione di estrazione delle informazioni;

3. Processa i dati usando algoritmi di topic model e salva l'output in formato leggibile.

Per eseguire il compito l'algoritmo è composto da diversi livelli, ognuno dei quali assolve a una specifica funzione.

L'obiettivo è quello di ottenere una procedura automatica e completa di classificazione dei contenuti della collezione di documenti analizzata.

L'output che desideriamo è dunque costituito da diversi cluster di parole (tanti quanti saranno i topic supposti latenti), ciascuno dei quali rappresenta un argomento centrale nella collezione. L'intero algoritmo, come specificato nei punti che esso eseguirà, è autonomo e l'intervento umano è ridotto al minimo. Una fase in cui quest'ultimo è indispensabile è certamente quella in cui si conferisce un titolo, o *label*, a ciascun cluster di parole: tale operazione non è certamente possibile per una macchina, ovvero per essa non è possibile trovare quella parola o quel concetto che sottende le parole in ciascun cluster estratte dai documenti.

Per intenderci, procediamo con un esempio: potremmo avere un cluster di parole come le seguenti:

```
giallo agrume aspro succo buccia ...
```

il computer avrà pure estratto con criterio queste parole dai documenti che ha rilevato, ma non sarà mai in grado di astrarre da questi concetti e conferire a questo cluster il label di 'Limone'.

Quest'ultima astrazione spetta ancora all'uomo.

4.1 Raccolta e storage dei dati

La raccolta dei file di testo avviene con Scrapy. Mediante esso si richiamano, dalla cartella del progetto iniziale, i file contenenti diversi sottomoduli, ciascuno dei quali va preimpostato indicando le specifiche del sito, dell'informazione da ricavare e del salvataggio da effettuare. Vediamo nel dettaglio i particolari principali di questo processo.

Supponiamo, ad esempio, di voler prelevare il testo della pagina iniziale di un sito di nostra scelta. Creiamo un progetto di lavoro dal nome *Example_ Name* avviando Scrapy dal prompt dei comandi e digitando

```
scrapy startproject Example_Name
```

Questo creerà una cartella con lo stesso nome del progetto con all'interno diversi file e cartelle con specifiche funzionalità: di essi ne analizziamo i più importanti. Il primo che prendiamo in esame ha nome *'items.py'*, che ha il ruolo di definire i campi da riempire con le informazioni, ovvero si imposterà la struttura del file che poi verrà salvato nel database. Ad esempio, supponendo di essere interessati ai titoli dei paragrafi ed ai testi presenti nella pagina analizzata, avremo un codice simile al seguente

```
class Example_Name(scrapy.Item):
    Titoli_paragrafi = Field()
    Contenuto_testi = Field()
...
```

Questo codice verrà richiamato nel file *'Example_ Name.py'* presente nella cartella *spiders*, la quale, come il nome suggerisce, rappresenta il cuore del nostro scraper. Nel suddeto file si specifica l'URL del sito da analizzare e il pezzo di codice HTML che affianca nell'impaginazione l'informazione desiderata. Una funzionalità molto interessante e utile di questo file è quella che consente di selezionare informazioni a più livelli di profondità nel sito in questione.

Come esempio, si pensi all'analisi dei quotidiani che andremo ad effettuare: la pagina principale di un quotidiano contiene molte notizie, ma nessuna per intero: per leggere il contenuto di una notizia occorre, generalmente, cliccare sul titolo e cambiar pagina, e dunque URL. L'indirizzo URL dell'articolo che si sta selezionando è chiaramente presente nel codice della pagina iniziale (in quanto cliccando su una specifica notizia siamo automaticamente trasportati alla pagina che la riguarda) e dunque basta reperire l'indirizzo specifico,

reindirizzare lo spider al nuovo indirizzo ed effettuare lo scraping sulla nuova pagina. Con questo metodo si può andare estremamente in profondità nel sito e prelevare tutta l'informazione senza per questo dover conoscere in anticipo tutti gli indirizzi dei contenuti, che, non è escluso, potrebbero modificarsi nel tempo. Con un'impostazione dinamica dello scraping questo problema è risolto.

Generalmente l'URL che conduce ad un contenuto specifico del sito, non è dissimile dall'URL del sito stesso: sovente il primo è un'estensione del secondo.

Un esempio di codice che, all'interno del file, esegue la suddetta operazione è il seguente:

```
def parse(self, response):
        links = list(set(response.xpath(xpath).extract()))
        for link in links:
            if link is not None:
                yield scrapy.Request(link,
callback=self.parse_page_Example_Name)
```

Come si nota, prima si estraggono i link 'suffissi' dei contenuti di interesse all'interno del sito e poi si aggregano questi ultimi al link della pagina principale.

A questo punto occorre impostare la procedura di salvataggio. La nostra scelta di database è MongoDB e tutte le impostazioni relative al salvataggio vanno inserite nel file *'pipelines.py'*. In esso si dànno istruzioni per inizializzare una diversa istanza per ogni nuova occorrenza da salvare e si imposta il filtro per evitare la presenza di duplicati. Quest'ultimo è un punto cruciale nel contesto di un'analisi, in quanto diventerebbe complesso riuscire a porvi rimedio durante le fasi successive.

Dell'intero codice, la parte fondamentale è il seguente stralcio:

```
 def process_item(self, item, spider):
        self.db[self.collection_name].update({'id': item['id']},
dict(item), upsert=True)
        return item
```

In particolare è l'opzione *upsert* che presiede all'inserimento di nuovi dati: nel codice appena visto, impostato sul valore *True*, esso non memorizza elementi che possiedano lo stesso identificativo di qualcuno già registrato.

Infine, nel file *'settings.py'*, si completano le ultime impostazioni specifiche relative alla porta di connessione del database e alla collection creata .

Dunque, in conclusione, la struttura dello scraper è ramificata e interconnessa: ogni modulo ha una specifica funzionalità che va impostata e ciascuno ne richiama un altro per ottenere il risultato finale.

Dopo questa fase, i file sono salvati in una collection la quale può essere estratta e manipolata anche esternamente al database. Le collection estratte sono salvate come file in formato JSON [1]: quest'ultimo ha una certa importanza per via della sua leggerezza e della sua flessibilità di utilizzo che coniuga un'impostazione semplice e leggibile sia per le macchine che per l'essere umano. Un esempio di file JSON è il seguente

```
{_id:
{id: 5a101c5d5fbf0004f5a4ade1},
     title:["Way of the Future, ..."],
 text: Nella Silicon Valley non c'è spazio soltanto...
 author:NP,
 date :["17 novembre 2017 "],
 link: http://www.corriere.it..."}
```

Una volta caricato il file di testo con tutte le informazioni in formato JSON siamo pronti per affrontare al fase di preprocessing.

4.2 Pulitura del file di testo

Le ragioni per cui è necessaria la pulitura di un file di testo sono già state in qualche misura elencate: in sintesi, in un approccio *bag-of-words* dove, cioè, la sola parola ha importanza, occorre eliminare le parole non connesse con i concetti latenti (come articoli, congiunzioni, preposizioni) e rendere indistinguibile agli occhi del calcolatore quelle parole che, seppur diverse, hanno lo stesso significato (come un verbo all'infinito con tutte le sue declinazioni). In sostanza si tratta di effettuare una operazione di *stopwords removing* ed una di *stemming*. Chiaramente queste analisi sono le principali, ma qualsiasi testo è suscettibile di diversi aggiustamenti, ognuno dei quali può in qualche misura migliorarne la comprensione.

Per prima cosa però, vanno eliminate alcuni caratteri o stringhe di caratteri che sono residui del linguaggio HTML che spesso filtrano durante la procedura di salvataggio del testo. Successivamente si conservano solo i caratteri alfabetici maiuscoli, minuscoli e accentati. Si eliminano numeri, segni di punteggiatura e tutte le lettere singole poiché queste ultime potrebbero essere o preposizioni, o congiunzioni o residui dell'impaginazione HTML (si pensi a \n, che indica andare a capo) . Il testo ottenuto viene, infine, ridotto tutto a caratteri minuscoli.

La procedura di rimozione delle stopwords è contenuta nella routine ***clean***.

Il passo iniziale è quello di richiamare il dizionario delle stopwords di *nltk*

[1] È un acronimo che sta per *JavaScript Object Notation*

```
stop_words = set(stopwords.words('italian'))
```

Potrebbe essere necessario aumentare l'estensione di questo dizionario predefinito, inserendo in esso nuove parole. Il codice per questa operazione è il seguente

```
nuove_parole = set(('colla', 'colle', 'sulle'...))
stopstop_words = set(stopwords.words('italian')) + nuove_parole
```

Una volta richiamata la funzione ***clean***, si opera con lo stemmer che va dapprima richiamato dal solito pacchetto *nltk*

```
stemmer = SnowballStemmer("italian") .
```

Da un input di testo iniziale come il seguente (tratto da una notizia di quest'anno)

```
Utilizzando i dati del Ministero dell'Istruzione si è lanciato
l'allarme: sono pochi i professori di Matematica, quindi le cattedre di
ruolo rimangono vuote. Il Problema si avverte soprattutto nella scuola media:
quest'anno si crea una voragine...
```

si otterrà come risultato della pulitura, il seguente output.

```
utilizz dat minister istruzion lanc allarm poch
prof matemat quind cattedr ruol rimang vuot problem
avvert soprattutt scuol med quest anno cre voragin...
```

che, sebbene sia fruibile per le macchine, dal punto di vista degli uomini è praticamente incomprensibile.

Onde evitare di ottenere un output formato da parole di cui non se ne riconosce il significato, è stata creata un altra routine di nome ***rebuild*** la quale associa a una radice semantica tutte le parole che in essa sono state trasformate dall'operazione di pulitura. Inoltre vengono contate le occorrenze delle singole parole originali. Un esempio di output della routine è il seguente

```
'comput' = [['computer', 10], ['computazionale', 2]]
```

volendo significare che, nel testo analizzato compare 10 volte la parola 'computer' e 2 volte la parola 'computazionale' e che, nel caso in cui la parola 'comput' sia rilevante nell'output finale, si può a ben ragione individuare quale sia l'origine più probabile di questa radice. Questo metodo non è immune dai casi di omonimia: non riesce a discriminare il significato che rivestono tali parole in un testo. Ad esempio, se la parola *stato* voglia significare *Nazione* o sia un participio passato.

Il testo così processato può finalmente essere analizzato dai modelli di topic extraction.

4.3 Fusione dei modelli e salvataggio

Il modello LDA è già stato discusso nell'ambito delle sue caratteristiche teoriche; l'algoritmo viene richiamato dalla libreria *MLlib* tramite *pyspark*. È all'LDA che spetta il processo del grosso dei file di testo, pertanto è in questa sede che occorre sperimentare i vantaggi della parallelizzazione che si ottengono con Spark. L'algoritmo dell'LDA estrae i principali concetti presenti in un testo ripulito, elencando un certo numero fissato a priori di parole per ciascun concetto. In sostanza abbiamo due parametri liberi: il numero dei topic, poniamo k, e il numero delle parole per ciascun topic, poniamo n. Il loro valore deve essere fissato prima di lanciare l'esecuzione e dipende fortemente dal tipo di file che si sta analizzando. Se ad esempio ci occupiamo di analizzare un racconto in formato testuale, il numero di topic k potrebbe essere elevato, viceversa, nel caso di un testo di breve estensione è più opportuno regolare il numero di topic su un valore più contenuto. Per quanto riguarda il parametro che indica il numero di parole per ogni topic, valgono in generale ragionamenti analoghi, anche se l'esperienza mostra un numero n compreso tra 10 e 20 generalmente consente di comprendere sufficientemente a fondo il concetto analizzato. È su base empirica che si effettua questa regolazione, detto *tuning*, dei parametri: occorre effettuare diverse prove per comprendere i valori ottimali nel caso specifico che si sta analizzando.

Nello specifico, le parti fondamentali del codice Spark sono le seguenti

```
data = sc.textFile(file_path).zipWithIndex().map( lambda cleanwords_idd:
       Row(idd = cleanwords_idd[1], cleanwords = cleanwords_idd[0].split(" ")))

docDF = spark.createDataFrame(data)
```

Con cui abbiamo creato un RDD. Il comando *textFile* legge il file di testo, ovvero il corpo dell'articolo, all'indirizzo *file_path* specifico, e indicizza i termini con il comando *zipWithIndex*, il quale associa ad ogni parola un numero (numeri uguali a parole uguali); infine, con la trasformazione *map* si crea una matrice Term-Document e il comando *createDataFrame* genera un dataframe a partire da *data*. Successivamente abbiamo

```
Vector = CountVectorizer(inputCol="cleanwords", outputCol="vectors")
model = Vector.fit(docDF)
result = model.transform(docDF)
corpus = result.select("idd", "vectors").rdd.map(lambda x_y: [x_y[0],
Vectors.fromML(x_y[1])]).cache()
```

ovvero inizializziamo il *CountVectorizer* che trasforma una collection di testo in un vettore di *token* e lo applichiamo al nostro dataframe. Il risultato è il modello di trasformazione che occorre al dataframe, un passaggio intermedio che permette di generare il *corpus*, che sarà l'input dell'LDA. Quest'ultimo

si ottiene come RDD selezionando con un'altra trasformazione gli input di interesse.

Infine si esegue

```
ldaModel = LDA.train(corpus, k = 2, maxIterations = 100, optimizer='em')
```

Da cui si ricavano i topic.

Il file completo di tutti questi topic raccolti viene passato all'ultimo blocco del codice, quello che applicherà la metodologia LSI, non prima però che venga effettuata un'ultima operazione.

Essa consiste nella rimozione di ulteriori categorie di parole. Per comprendere la necessità di quest'ulteriore scrematura, dobbiamo fare qualche passo avanti. Ci aspettiamo che l'output finale sia composto da diversi cluster di parole, ciascuno dei quali può simboleggiare un concetto importante nella collezione. Ciascun termine proviene da da un cluster di parole ottenuto mediante la precedente operazione con l'LDA.

Si nota che nei cluster sopravvivono parole come 'quas' (che sta per 'quasi') o 'fa' (dal verbo 'fare'). Queste categorie di forme verbali e avverbi sono molto comuni a tutti i documenti e dunque è plausibile che molti cluster finali siano 'contaminati' dalla presenza di queste parole che non portano con sé significato. Si immagini ad esempio un cluster di soli avverbi: quale contenuto potrebbe significare? Eppure, si sperimenta, che cluster di questo tipo tendono effettivamente a formarsi dato il numero di occorrenze elevato in molti topic di queste categorie di parole. Ecco perché queste parole vanno eliminate con l'uso di un'ulteriore operazione di *stopwords removing* eseguita con un dizionario *ad hoc*. A questo punto si può invocare il modello LSI per la clusterizzazione finale: l'algoritmo opererà con le modalità già descritte il precedenza.

L'output sarà simile al seguente:

```
Topic 1:

1) stat
[['stato', 752], ['stata', 419], ['stati', 257], ['state', 98],
['statuto', 3]]

2) part
[['parte', 366], ['partito', 95], ['partire', 57], ['partita', 53],
['parti', 33], ['partiti', 24], ['partite', 22], ['partendo', 8], ...]

3) ital
[['italia', 376], ['italiano', 104], ['italo', 4], ['italici', 2],
['italico', 2], ['italiche', 1]]

...
```

```
Topic 2:

1) ministr
[['ministro', 95], ['ministri', 29], ['ministra', 17]]

2) paes
[['paese', 192], ['paesi', 75]]

3) lavor
[['lavoro', 301], ['lavoratori', 59], ['lavorare', 48], ['lavori', 43],
['lavorato', 36], ['lavora', 34], ['lavorando', 20], ['lavorano', 15],
['lavorava', 10], ['lavoravano']]

...

Topic 3:

1) part
[['parte', 366], ['partito', 95], ['partire', 57], ['partita', 53],
['parti', 33],
['partiti', 24], ['partite', 22], ['partendo', 8], ...

2) stat
[['stato', 752], ['stata', 419], ['stati', 257], ['state', 98],
['statuto', 3]]

3) rapport
[['rapporto', 79], ['rapporti', 50]]

...
```

reso intelligibile grazie alla routine ***rebuild*** descritta sopra. Anche in questo caso a ciascuna parola è associata una probabilità, con il medesimo significato di rilevanza con il topic come nell'output precedente.

È necessario, infine, dare una giustificazione della procedura così impostata, ovvero della fusione dei modelli e dell'ultima operazione di pulitura, il cui posizionamento all'interno dell'algoritmo può sembrare un pò anomalo.

Partiamo da quest'ultimo punto: perché eseguire lo stopwords removing in due diversi momenti del codice? Ci sono due motivazioni: la prima consiste nel fatto che prima di passare al vaglio dell'LSI, le parole eliminate potrebbero essere effettivamente cariche di significato per l'LDA e in un cluster di parole potrebbero contribuire a chiarificare il significato di un topic.

Rammentiamo che ogni procedura di riduzione del documento tende ad eliminare parte dell'informazione e dunque, quando non è necessaria, è buona norma non abusarne. Il secondo motivo è più pratico: si è già parlato della carenza di dizionari per le analisi testuali della lingua italiana. Dovendo far fronte a parole che hanno già subìto il processo di stemming, si può più agevolmente costruire un dizionario *ex novo*.

Per quanto riguarda la motivazione della scelta di queste due metodologie e della fusione nel preciso ordine mostrato, dobbiamo fare delle riflessioni.

Innanzitutto come estrattore dei topic di ciascun documento, era scontato che usassimo un modello di tipo probabilistico, per le ragioni concernenti i limiti di un modello deterministico e non generativo di cui si è discusso in precedenza. Una volta terminata questa fase, si può passare ad un modello come l'LSI che, in modo particolare, riesce a rilevare concetti come la sinonimia, che sono rilevanti per una classificazione ad un secondo stadio; inoltre come algoritmo è estremamente rapido.

Ci si può chiedere, a questo punto, se fosse stato possibile eseguire due volte l'LSI o due volte l'LDA invece che fondere le due metodologie. In generale è perfettamente lecito impostare in questo modo l'algoritmo che, non è escluso, potrebbe fornire risultati degni di nota. Tuttavia nel caso di un'applicazione a due stadi dell'LSI mancherebbe completamente la dimensione probabilistica, ovvero non si darebbe conto della struttura generativa del singolo documento. Nel caso opposto, invece, quello che contempla un'applicazione ripetuta dell'LDA, dobbiamo ipotizzare di ricavare diversi cluster di concetti da una fusione di tutti i topic provenienti dai documenti. Nel secondo stadio, dunque, avremmo perso completamente la struttura generativa che contraddistingue un testo in linguaggio naturale, dovendo noi processare esclusivamente liste di parole chiave per ciascun topic. Ne risulterebbe un processo più dispendioso e meno congeniale alle effettive operazioni di analisi che si svolgono. È altresì chiaro da questi ragionamenti per quali motivi non è consigliato scambiare le due metodologie in questa analisi, ovvero eseguire prima l'LSI e poi l'LDA.

Un ultimo punto importante è chiarire per quale motivo non abbiamo adoperato un altro algoritmo generativo come il pLSI al posto dell'LDA. Anche questa modifica sarebbe lecita, tuttavia abbiamo visto quale sia la limitazione del pLSI a livello di collection. Nel nostro caso, avendo usato l'LDA, la procedura impostata è più facilmente generalizzabile e possiamo modificarla opportunamente e applicarla, in teoria, anche a collection di collection, senza tener conto delle restrizioni che il pLSI imporrebbe.

La figura 4.1 riassume in uno schema pratico l'intero sistema distribuito

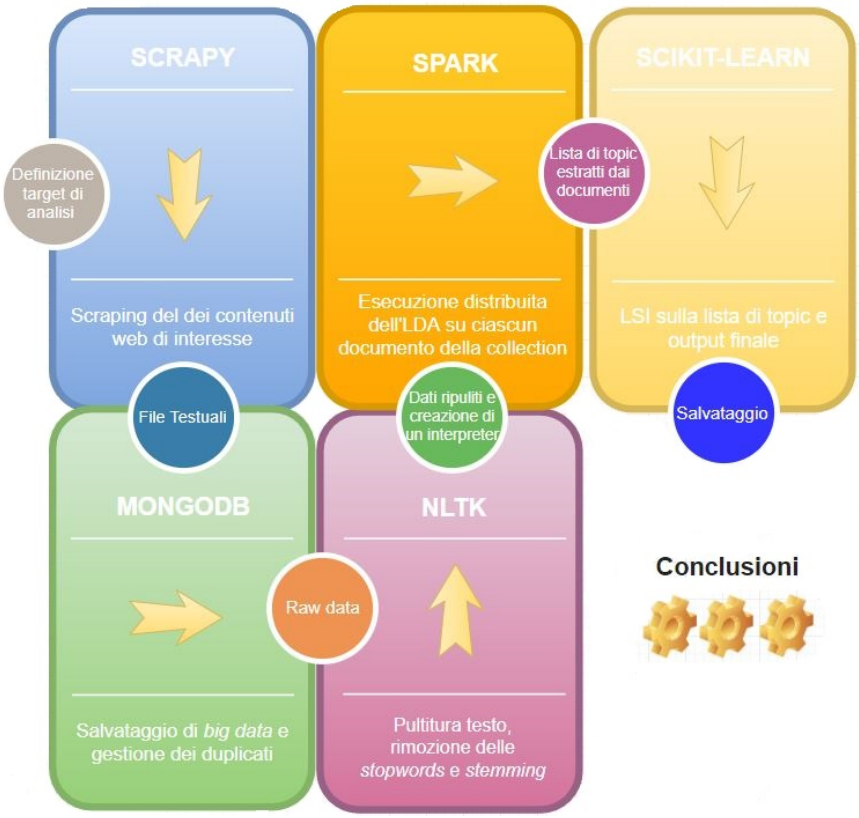

Figura 4.1: Schema riassuntivo

Capitolo 5

Presentazione del caso di studio

A conclusione del sistema distribuito sviluppato, presentiamo il seguente studio.

I testi analizzati, come avevamo anticipato, sono costituiti da articoli di giornale e l'obiettivo è quello di operare una classificazione non supervisionata dei contenuti, ovvero di enucleare i temi principali di ciascuna testata giornalistica analizzata durante la finestra temporale di raccolta notizie.

Abbiamo selezionato 5 testate: la scelta è in gran misura arbitraria, tuttavia è necessario che due requisiti siano soddisfatti:

1. devono pubblicare in rete gli articoli principali ed esse devono essere fruibili gratuitamente per l'utente;

2. deve trattarsi di testate di rilievo nazionale, per avere a disposizione un gran numero di articoli su cui lavorare.

La nostra scelta è costituita dalle seguenti testate: *Repubblica, Il Corriere della Sera, La Stampa, Il Giornale, Fanpage.*

Quest'ultima è pubblicata esclusivamente in rete, è interamente gratuita e costituisce un caso interessante da confrontare alle altre testate che sono più antiche per formazione e sono state pubblicate per la maggior parte del tempo su carta stampata.

I siti da cui sono stati tratti gli articoli, non sono quelli principali dei singoli giornali, bensì quelli che contengono i Feed RSS, i cui indirizzi sono rispettivamente [RepRSS], [CorRSS], [LaSRSS], [IlGRSS], [FanRSS].

La scelta di considerare i Feed RSS è giustificata dalla semplicità maggiore con cui si può effettuare il parsing della pagina principale. In più le notizie pubblicate sulla mainpage di ciascun quotidiano non differiscono da quelle che si trovano sotto forma di Feed.

È importante considerare che alcune testate possiedono sezioni per diverse tematiche (cronaca sportiva, rosa, finanza), nel nostro caso ci siamo

limitati allo scraping della pagina principale, ovvero della vetrina notizie che ogni giornale compone, supponendo trattarsi di quelle di maggior rilievo per quella testata.

Su MongoDB gli articoli vengono salvate in diverse collection, ciascuna relativa ad una testata. I file salvati hanno comunque la stessa struttura, costituita da

1. *ID*: viene automaticamente assegnato ad un nuovo ingresso in memoria;
2. *Title*: in cui viene salvato il titolo dell'articolo di giornale;
3. *Resume*: un riassunto, se presente dell'articolo. A volte, quando presenti, aggregano occhiello e sommario;
4. *Date*: la data di pubblicazione dell'articolo;
5. *Author*: l'autore, se presente, dell'articolo;
6. *Shares*: Il numero di condivisioni che un articolo possiede sui social;
7. *Tags*: etichette che classificano l'articolo in categorie;
8. *Text*: il testo dell'articolo;
9. *Link*: il link originale dell'articolo;
10. *Newspaper*: il giornale su cui è comparsa la notizia.

Non tutti i campi saranno utilizzati nel prosieguo, la presenza di essi è giustificata da un'impostazione generale con cui si è condotto lo studio: alcuni parametri possono avere un interesse per diversi studi, anche diversi da questo, per cui, questo algoritmo può riadattarsi e generalizzarsi anche a diverse situazioni. Inoltre, talvolta, alcuni campi (come *Author, Tags, Resume*) possono rimanere vuoti perché non presenti nel testo originale.

Un esempio di articolo salvato è il seguente

```
{ ID:
{ id:5a101c445fbf0004f5a4ace5},
 title : In Cina il primo trapianto di testa del mondo... ,
 text: Il primo trapianto di testa del mondo è stato ...,
 newspaper: Fanpage,
 resume:  L'operazione sperimentale è stata condotta con successo ...,
 author: D. F.,
 date: 17 novembre 2017
 shares: 321,
 tags: Medicina, Mondo ...,
 link: https://www.fanpage.it/in-cina-il-primo...}
```

Di esso accorperemo le voci *title, text, tags, resume* a formare un'unica stringa di testo che rappresenterà il nostro documento da analizzare.

Il periodo di raccolta, per tutti i quotidiani, è partito il 17 Novembre 2017 ed è terminato il 16 Dicembre dello stesso anno.

Osserviamo che il periodo di raccolta non è esteso; come abbiamo anticipato, le conclusioni che possiamo trarne sono solo indicative: uno studio rigoroso, in generale, dovrebbe tener conto di un periodo di raccolta più ampio.

In questo lasso di tempo abbiamo, comunque, raccolto e analizzato oltre duemilacinquecento articoli, ovvero mediamente 500 articoli per testata. Se intendessimo analizzare dieci testate su un periodo di osservazione lungo un anno, dovremmo gestire circa 60 mila articoli: in quest'ottica, il sistema distribuito, la scelta del database e, in generale, l'impostazione di tutto l'algoritmo sono pienamente giustificati.

Lo scraper è stato avviato ogni giorno durante il mese di raccolta. Su ciascun sito di sovente accadeva di trovare articoli risalenti al giorno precedente, per cui le impostazioni di salvataggio contemplavano la possibilità di avere dei doppi e si evitava di salvarli confrontando il titolo (cfr. Cap. 2): di due articoli con lo stesso titolo viene conservato il più recente, in modo da aggiornare alcuni contenuti dinamici come il numero di condivisioni sui social.

In generale, il numero degli articoli è un parametro variabile che dipende da diversi fattori. Esso, insieme ad altri valori indicativi circa l'analisi, sarà indicato ogni volta che un output sarà esposto.

L'intera procedura è stata eseguita su un computer con le seguenti caratteristiche:

- Processore: Intel Core i3
- RAM: 4.00 GB
- CPU: 1.80 GHz

Chiaramente le specifiche temporali sono relative al tipo supporto hardware che si utilizza, per cui, ad esempio, con il doppio o il quadruplo di *core* con medesime caratteristiche, il tempo di esecuzione dimezza, diventa un quarto, ecc.

5.1 Output finale

Prima di passare ad illustrare l'output, facciamo un breve riassunto dell'impostazione del sistema, elencando i punti principali e descrivendo le operazioni eseguite durante ciascuna fase per il completamento dello studio finale:

1. *Fase di Scraping*: durante il periodo di raccolta, vengono prelevati i contenuti testuali dai siti di Feed RSS prescelti;

2. *Fase di memorizzazione*: che va di pari passo allo scraping. Tutte le informazioni vengono salvate nel formato mostrato in precedenza e si presta attenzione ai possibili doppi estratti confrontando il titolo;

3. *Fase di Pulitura*: in cui si trasferisce il contenuto di ciascuna collection nel database in un file di tipo JSON e lo si sottopone ad una procedura di omogeneizzazione dei caratteri e ripulitura da segni grafici diversi dal quelli alfabetici (segni di interpunzione, numeri, *backslash*, ecc.), in modo da eliminare anche il residuo del linguaggio HTML di ciascuna notizia;

4. *Fase di NLP*: in cui si applica una procedura di rimozione delle *stopwords* e si esegue lo *stemming* delle parole rimaste;

5. *LDA con Spark*: in cui si richiama la console Spark e si esegue l'algoritmo LDA sugli articoli. Si definisce un corpus costituito dai topic estratti e si applica una nuova rimozione delle *stopwords*;

6. *LSI sul corpus*: in cui si esegue l'LSI e si ottiene il risultato finale.

7. *Fase di salvataggio*: in cui viene salvato l'output.

In particolare, riguardo ai penultimi ultimi due punti, è necessario specificare i parametri utilizzati: nella fase 5 si sono estratti 2 topic composti da 12 parole per ciascun articolo; nella fase 6 si sono estratti 15 topic composti da 12 parole. Il *tuning* dei parametri è stato stabilito sulla base di diverse prove empiriche.

L'output originale di ciascuna prova è lungo decine di pagine e sarebbe scomodo riproporlo per intero: occorre un modo efficiente di rappresentarlo. Un esempio di risultato in forma originale è il seguente

```
Topic X:
1) vit
[['vita', 60]]

2) prezz
[['prezzo', 12], ['prezzi', 5]]

3) parl
[['parlare', 14], ['parlato', 12], ['parlamento', 11],
['parla', 10], ['parliamo', 5], ['parlano', 4], ...

4) stat
```

```
[['stato', 221], ['stati', 97], ['stata', 88], ['state', 32], ['statuto', 1]]
```

Come si nota ci sono 4 tipologie di radici stemmizzate:

1. Quelle che si associano senza dubbio alla parola originaria, come *vit*: scriveremo semplicemente *vita* senza specificare la classe;

2. Quelle che si riferiscono a diverse parole, ma il cui significato è il medesimo, come *prezz*: scriveremo in questo caso, l'occorrenza più frequente e cioè *prezz**o*** che intenderemo come rappresentante della classe;

3. Quelle che si riferiscono a diverse parole, ma che hanno significati differenti, come *parl*: scriveremo *parl**are***/*parl**amento***;

4. Quelle la cui parola originaria resta comunque dubbia, come *stat*: scriveremo *stat**o***, ma se l'occorrenza si riferisca al concetto di Stato inteso come nazione o al participio passato del verbo essere non è dato saperlo. Purtroppo questo è un problema intrinseco della procedura che non potrà essere risolto in questa sede.

In presenza di occorrenze molto numerose ignoreremo quelle sporadiche anche se il significato differisce (come nel caso di *stat**o***, a cui non abbiamo affiancato la parola *statuto* perché compare una volta sola). A patto di sacrificare la visione del numero di occorrenze, che segnaleremo se degno di nota, potremo in questo modo visualizzare per intero e senza difficoltà il risultato dell'analisi.

A questo punto dovremmo verificare che l'output del nostro sistema è effettivamente coerente: proviamo con un caso controllato con la collection di nome *Sport* che raccoglie 50 articoli sportivi recenti relativi al campionato calcistico italiano (Serie A), in particolare degli ultimi eventi sportivi[1]. Dato il numero di articoli ridotto, solo in questo caso abbiamo 3 topic finali di 10 parole:

1. **Topic 1**: port**a**, inter, spalletti, fiorentina, simeone, chiesa, icardi, allen**atore**, squadr**a**, ultim**e**;

2. **Topic 2**: chi**evo**, ser**ie**, udines**e**, radovanovic, tomovic, allen**atore**, vittor**ia**, oddo, part**ita**, napoli;

3. **Topic 3**: allen**atore**, gruppo, port**a**, segn**a**, lavor**o**, ultim**e**, meret, salvezz**a**, adess**o**, oddo.

I cui temi sono perfettamente in linea con le aspettative.
Passiamo agli output del nostro studio.

[1] Settimana dall'1 al 6 Gennaio 2018

5.1.1 Repubblica

Statistiche[2]

```
Totale articoli: 448
Totale parole: 125038
Media parole per articolo: circa 279
Mediana parole per articolo: 257
Massimo numero di parole per articolo: 1032
Minimo numero di parole per articolo: 20
```

L'output è il seguente:

1. **Topic 1**: stat**o**, lavor**o**/lavor**atori**, part**e**/part**ito**/part**ire**, president**e**, mil**a**/mil**ano**, donn**e**, ital**ia**, figl**i**, politic**a**, eur**o**, bambin**i**, sindac**o**/sindac**ati**;

2. **Topic 2**: part**e**/part**ito**/part**ire**, lavor**o**/lavor**atori**, ital**ia**, coalizion**e**, politic**a**, vot**o**, social/sociale, milion**i**, governo, success**ivo**, poss**ono**, person**e**;

3. **Topic 3**: president**e**, polit**ica**, vot**o**, part**e**/part**ito**/part**ire** , alcun**i**, dem, deput**ato**, coalizion**e**, poss**ono**, micciché, parl**are**/parl**amento**, maggior**e**/maggior**anza**;

4. **Topic 4**: president**e**, lavor**o**/lavor**atori**, alcun**i**, parl**are**/parl**amento**, maggior**e**/maggior**anza**, banc**a**, deput**ato**, vot**o**, dem, eur**o**, micciché, commissione;

5. **Topic 5** mil**a**/mil**ano**, eur**o**, pag**are**, bambin**i**, rom**a**, success**ivo**, figl**i**, comun**e**/ comun**icazione**/comun**ista**, procur**a**, condivid**i**, person**e**, tribunal**e**;

6. **Topic 6**: ital**ia**, eur**o**, giovan**i**, social/sociale, paes**e**, med**ia**/med**io**, pag**are**, arrest**ato**, milion**i**, omicid**i**, person**e**, poliz**ia**

7. **Topic 7**: mil**a**/mil**ano**, part**e**/part**ito**/part**ire**, eur**o**, coalizion**e**, stat**o**, unit**i**, occup**azione**, dett**o**, pag**are**, ex, figl**i**, punt**o**;

8. **Topic 8**: person**e**, condizion**i**, migrant**i**, ricerc**a**, milion**i**, rom**a**, san, success**ivo**, unit**i**, rohingya, francesc**o**, sal**ute**/sal**uto**;

9. **Topic 9**: francesc**o**, rohingya, **e**/ comun**icazione**/comun**ista**, pap**a**, bambin**i**, san, paes**e**, bangladesh, myanmar, birman**i**, autor**ità**, pontef**ice**;

[2]Per *Totale parole* si intende il conteggio delle parole stemmizzate, ovvero il conteggio che viene effettuato quando gli articoli hanno già subito il processo di pulitura e di rimozione delle stopwords.

10. **Topic 10**: condivid**i**, part**e**/part**ito**/part**ire**, procur**a**, repubbl**ica**, giornal**isti**, commission**e**, dichiar**ato**/dichiar**azioni**, indagin**i**, legg**e**/legg**ere**, ex, eur**o**, libr**i**

11. **Topic 11**: donn**e**, mar**ia**/ mar**e**/ mar**ito**, rom**a**, lavor**o**/lavor**atori**, de, pap**a**, colp**ito**, francesc**o**, denunc**ia**, mort**e**, rohingya, parol**e**;

12. **Topic 12**: procur**a**, sindac**o**/sindac**ati**, indagin**i**, ricerc**a**/ricerc**atori**, chiest**o**, denunc**ia**, assoc**iazione**, torino, legg**e**/legg**ere**, part**e**/part**ito**/part**ire**, person**e**, comun**e**/ comun**icazione**/comun**ista**;

13. **Topic 13**: forz**a**/forz**e**, rom**a**, poliz**ia**, cap**o**/cap**ire**, presidentee, scuol**a**, mil**a**/mil**ano**, present**ato**/present**e**, presid**io**, altern**anza**, attent**ato**, partecip**ato**;

14. **Topic 14**: success**ivo**, ricerc**a**, precedent**e**, trov**ato**, vit**a**, ben**e**, numer**o**, mond**o**, univers**ità**, apert**o**, equipagg**io**, amant**e**;

15. **Topic 15**: pag**are**, eur**o**, pap**a**, lavor**o**, giudic**e**, rohingya, omicidio, vit**a**, francesc**o**, scuol**a**, alcun**i**, bar**i**;

I tempi di esecuzione sono stati i seguenti:

- LDA con Spark: 1350 secondi;
- LSI: 4 secondi

L'intero processo di elaborazione è durato circa 25 minuti.

5.1.2 Il Corriere

Statistiche

```
Totale articoli: 924
Totale parole: 224121
Media parole per articolo: circa 243
Mediana parole per articolo: 223
Massimo numero di parole per articolo: 1104
Minimo numero di parole per articolo: 15
```

L'output è il seguente:

1. **Topic 1**: stat**o**, part**e**/part**ito**/part**ire**/ part**ita**, lavor**o**, mil**ano**/mil**a**, donn**a**, ital**ia**, ragazz**i**, figl**io**, person**e**, polit**ica**, port**a**, parl**a**/parl**amento**;

2. **Topic 2**: stat**o**, donn**a**, ragazz**i**, poliz**ia**, accus**a**, uom**o**, omicid**io**, figl**io**, procur**a**, ferm**ato**, carabinieri, ucciso;

3. **Topic 3**: part**e**/part**ito**/part**ire**/ part**ita**, ital**ia**, polit**ica**, pd, renz**i**, parl**a**/parl**amento**, vot**o**, berlusconi, presidente, ser**ie**/ser**a**, rom**a**, stat**o**

4. **Topic 4**: lavor**o**, part**e**/part**ito**/part**ita**, pd, governo, renz**i**, pens**are**, legg**e**/legg**ere**, vot**o**, ministr**o**, stat**o**, polit**ica**, presidente;

5. **Topic 5** mil**ano**/mil**a**, part**e**/part**ito**/part**ita**, lavor**o**, eur**o**, stat**o**, centr**o**, presidente, novembre, italian**a**, governo, cost**a**, vend**ita**;

6. **Topic 6**: ital**ia**, paese, rom**a**, presidente, stat**o**, ministr**o**, donn**a**, forz**a**/forz**e**, pubblic**o**/pubblic**a**, poss**ono**, success**o**, sub**ito**;

7. **Topic 7**: figl**io**, vit**a**, donn**a**, mil**ano**/mil**a**, parl**a**/parl**amento**, renz**i**, italian**a**, polit**ica**, berlusconi, volo/vol**uto**, pd, numero;

8. **Topic 8**: parl**a**/parl**amento**, polit**ica**, renz**i**, berlusconi, dett**o**, volo/vol**uto**, pens**a**, person**e**, nome, pd, milion**i**, leader;

9. **Topic 9**: accus**a**, ser**ie**/ser**a**, ragazz**i**, enne[3], poliz**ia**, ex, sessuale, figl**io**, giudice, rapport**o**, iniz**o**, part**e**/part**ito**/part**ire**/part**ita**;

10. **Topic 10**: ragazz**i**, donn**a**, trov**ato**, violenz**a**, giovani, ital**ia**, parl**a**/parl**amento**, auto, polit**ica**, scuol**a**, ser**ie**/ser**a**, amici;

11. **Topic 11**: ser**ie**/ser**a**, vit**a**, stat**o**, mil**ano**/mil**a**, ultimi, napoli, italian**a**, classif**ica**, inter, pass**ato**/pass**o**, bene, gioco;

12. **Topic 12**: scuol**a**, volo/vol**uto**, ragazz**i**, ital**ia**, pubblic**o**/pubblic**a**, pens**are**, figl**io**, stat**o**, sent**ito**, già, prov**a**, insegn**ante**

13. **Topic 13**: legg**e**/legg**ere**, figl**io**, presidente, milion**i**, trov**o**, governo, commissione, bilanc**io**, morte, economi**a**, parl**a**/parl**amento**, genitori;

14. **Topic 14**: vit**a**, legg**e**/legg**ere**, pass**ato**, milion**i**, ragazz**i**, governo, strad**a**, mett**ere**, bilanc**io**, aut**o**, morte, giovani

15. **Topic 15**: port**a**, figl**io**, morte, lavor**o**, auto, ex, person**e**, bene, madre, eur**o**, poss**ono**, feriti;

I tempi di esecuzione sono stati i seguenti:

- LDA con Spark: 2804 secondi;
- LSI: 8 secondi

L'intero processo di elaborazione è durato circa 47 minuti.

[3]Deriva dall'uso di scrivere, ad esempio, *16enne* in luogo di *sedicenne* e così via...

5.1.3 La Stampa

Statistiche

```
Totale articoli: 176
Totale parole: 47229
Media parole per articolo: circa 268
Mediana parole per articolo: 253
Massimo numero di parole per articolo: 1077
Minimo numero di parole per articolo: 62
```

L'output è il seguente:

1. **Topic 1**: stat**o**, part**e**/part**ito**/part**ire**, ital**ia**, mil**a**/mil**ano**, paes**e**, governo, eur**o**, milion**i**, pd, commissione, russ**ia**, città;
2. **Topic 2**: part**e**/part**ito**/part**ire**, governo, pd, renz**i**, commissione, decisione, ue, vot**o**, sinistr**a**, accord**o**, may, pisapia;
3. **Topic 3**: ue, paes**e**, accord**o**, uniti, may, decisione, europe**a**, regno, britann**ico**, unione, russ**ia**, merc**ato**;
4. **Topic 4** eur**o**, ue, europe**a**, milion**i**, may, regno, accordo, uniti, negoz**iati**, stato, britann**ico**, premier;
5. **Topic 5**: mil**a**/mil**ano**, ital**ia**, men**o**, lavor**o**, paese, figl**io**, diminu**zione**, rispett**o**, piazz**a**, pag**are**, alcuni, chiara;
6. **Topic 6**: ital**ia**, russ**ia**, piazz**a**, pd, mosc**a**, vot**o**, sinistr**a**, paese, città, usa, palazzo, regole;
7. **Topic 7**: paese, figl**io**, pd, parl**are**/parl**amento**, vit**a**, prezz**o**, alcuni, tagli, dett**o**, russ**ia**, ius, legge;
8. **Topic 8**: città, pass**ato**, piazz**a**, torino, alcuni, part**e**/part**ito**, san, donne, ospedale, mondo, palazzo, appendino[4];
9. **Topic 9**: milioni, città, paese, mondo, ultimi, capitale, alcuni, parco, sir**ia**, dollari, punto, prezzo;
10. **Topic 10**: persone, parl**are**/parl**amento**, lavoro, altro, robot, dett**o**, vol**eva**/voli, conto, governo, pass**ato**, sera/ser**ie**, san;
11. **Topic 11**: presidente, san, mil**a**/mil**ano**, pass**ato**, piazz**a**, acqua, berlusconi, ogni, ex, certo, sera/ser**ie**, sir**ia**;
12. **Topic 12**: governo, sir**ia**, uniti, possibil**ità**, comune, ottobre, unico, nord, colp**ito**, prezzo, campo, centro;

[4]Riferito a Chiara Appendino, sindaco di Torino.

13. **Topic 13**: parl**are**/parl**amento**, bambin**i**, mond**o**, persone, città, italian**a**, milion**i**, cont**o**, lavor**o**, punt**o**, poch**hi**, giovan**i**;

14. **Topic 14**: port**ato**, figl**io**, dett**o**, renz**i**, parl**are**/parl**amento**, madr**e**, famigl**ia**, piazz**a**, europe**a**, affid**ata**, organizz**azione**, ue

15. **Topic 15**: eur**o**, dett**o**, piazz**a**, russ**ia**, tagl**i**, san, donn**e**, mosc**a**, fond**o**, figl**io**, renz**i**, acqua.

I tempi di esecuzione sono stati i seguenti:

- LDA con Spark: 525 secondi;
- LSI: 2 secondi

L'intero processo di elaborazione è durato circa 9 minuti.

5.1.4 Il Giornale

Statistiche

```
Totale articoli: 186
Totale parole: 40967
Media parole per articolo: circa 220
Mediana parole per articolo: 176
Massimo numero di parole per articolo: 873
Minimo numero di parole per articolo: 40
```

L'output è il seguente:

1. **Topic 1**: stat**o**, part**e**/part**ito**, pd, ital**ia**, punt**o**, eur**o**, figl**ia**, lavor**o**, renz**i**, person**e**, pass**ato**, berluscon**i**;

2. **Topic 2**: pd, part**e**/part**ito**, renz**i**, matte**o**, pers**o**, poss**ono**, pass**ato**, leader, elezion**i**, centrodestra, lavor**o**, ius

3. **Topic 3**: ser**ie**/sera, punt**o**, mil**a**/mil**ano**, rom**a**, de, dett**o**, pag**are**, vip, sub**ito**, berluscon**i**, squadr**a**, camp**o**

4. **Topic 4** punt**o**, ital**ia**, ultim**a**, ogni, mar**ia**/mar**e**/mar**ito**, gioc**o**, nonost**ante**, davver**o**, berluscon**i**, bal**o**, talent**o**, europe**o**

5. **Topic 5**: part**e**/part**ito**, pass**ato**, qualc**he**, cert**o**, aul**a**, numer**o**, pens**a**, depress**i**, deput**ati**, padr**e**, sent**e**, ser**ie**/ser**a**;

6. **Topic 6**: lavor**o**, spieg**a**, leader, racconta, famigl**ia**, tass**e**, paes**e**, fragil**ità**, plag**iata**, sett**e**, showgirl, person**e**;

7. **Topic 7**: punt**o**, present**e**, provinc**ia**, davant**i**, vien**e**, tratt**a**, suon**o**, squadr**a**, già, alto, bolzano, entrat**e**;

8. **Topic 8**: alcun**i**, papa, cardinal**e**, chies**a**, port**a**, magic**a**, mueller, ser**ie**/ser**a**, cerc**a**/cerc**hio**, scism**a**, berlusconi, vend**ita**;
9. **Topic 9**: present**e**, spieg**a**, provinc**ia**, entrat**e**, ital**ia**, ser**ie**/ser**a**, alto, bolzano, riport**a**, numer**o**, riguard**a**, viv**ere**;
10. **Topic 10**: provinc**ia**, giovan**e**, dat**o**/dat**a**, pubblic**o**, spos**ato**, bell**a**, alto, bolz**ano**, giudic**e**, ragazz**i**, famigl**ia**, rodriguez;
11. **Topic 11**: figl**ia**, padr**e**, pd, music**a**, cecilia, dev**e**, uomo, suon**o**, violin**o**, vien**e**, violenz, ignazio
12. **Topic 12**: berlusconi, viene, music**a**, ital**ia**, vot**o**, rom**a**, port**a**, person**e**, suon**o**, violin**o**, silv**io**, però;
13. **Topic 13**: rom**a**, scuol**a**, numer**o**, riport**a**, giovan**e**, napol**i**, bambin**i**, spos**ato**, nulla, stor**ia**, figl**ia**, davant**i**
14. **Topic 14**: figl**ia**, spos**ato**, giovan**e**, ragazz**i**, famigl**ia**, pd, padr**e**, dat**o**, entrat**e**, messagg**io**, consigl**io**, cent**o**;
15. **Topic 15**: person**e**, nott**e**, provinc**ia**, poliz**ia**, alcun**i**, part**e**/part**ito**, davanti, spieg**are**, silv**io**, novembre, ogni, alto.

I tempi di esecuzione sono stati i seguenti:

- LDA con Spark: 596 secondi;
- LSI: 4 secondi

L'intero processo di elaborazione è durato circa 10 minuti.

5.1.5 Fanpage

Statistiche

```
Totale articoli: 920
Totale parole: 181466
Media parole per articolo: circa 197
Mediana parole per articolo: 176
Massimo numero di parole per articolo: 1230
Minimo numero di parole per articolo: 60
```

L'output è il seguente:

1. **Topic 1**: stat**o**, donn**a**, figl**io**, uom**o**, mort**e**, enne, ragazz**a**/ragazz**o**, poliz**ia**, bambin**i**, trov**ato**, vittim**a**, arrest**ato**;
2. **Topic 2**: ital**ia**, part**e**/part**ito**/part**ire**, pd, polit**ica**, renz**i**, elezion**i**, forz**a**/forz**e**, president**e**, nord, eur**o**, leader, prossim**e**;

3. **Topic 3**: figl**io**, donn**a**, bambin**i**, genitor**i**, mar**it**o/mar**ia**, padr**e**, condann**ato**, pd, abus**i**, mamm**a**, cap**o**,cap**ire**, polit**ica**;

4. **Topic 4** nord, neve, donn**a**, figl**io**, temperatur**e**, piogg**e**, centr**o**, region**i**, sud, fredd**o**, metr**i**, cal**o**;

5. **Topic 5**: lavor**o**/lavor**atori**, bambin**i**, figl**io**, port**ato**, vit**a**, medic**i**, genitor**i**, azi**enda**, person**e**, ultim**o**, milion**i**, scuol**a**;

6. **Topic 6**: enne, uom**o**, giovan**e**, ragazz**a**, vittim**a**, carabinier**i**, racconta**to**, denunc**ia**, accus**a**, condann**a**, sessuale, ex

7. **Topic 7**: donn**a**, lavor**o**, fer**ite**, incident**e**, person**e**, poliz**ia**, ore, dett**o**, mogl**ie**, present**ato**/present**i**, legg**e**/legg**ere**, colp**ito**;

8. **Topic 8**: ragazz**a**, donn**a**, vit**a**, giovan**a**, madr**e**, figl**io**, vittim**a**, famigl**ia**, decis**o**, trov**ato**, indagin**e**, continu**a**;

9. **Topic 9**: bambin**i**, stat**o**, bimb**o**, racconta**to**, scuol**a**, violenz**a**, genitor**i**, denunc**ia**, donn**a**, giovan**e**, vittim**a**, sessuale;

10. **Topic 10**: poliz**ia**, fer**ite**, person**e**, incident**e**, bimb**o**, bambin**i**, figl**io**, ragazz**a**, ore, fuoc**o**, mattin**a**, genitor**i**

11. **Topic 11**: ragazz**a**, stat**o**, lavor**o**, uom**o**, figl**io**, decis**o**, ex, presidente, mar**it**o/mar**ia**/mar**e**, vit**a**, poliz**ia**, eur**o**;

12. **Topic 12**: bosc**hi**, banc**a**, presidente, etrur**ia**, dichiar**ato**, ex, oper**a**/oper**azione**, alcun**i**, stat**o**, legg**e**/legg**ere**, incontr**o**, person**e**;

13. **Topic 13**: ragazz**a**, dett**o**, vit**a**, bambin**i**, medic**i**, port**ato**, pd, ospedale, part**e**/part**ito**/part**ire**, elezion**i**, genitor**i**, renz**i**

14. **Topic 14**: trov**ato**, part**e**/part**ito**/part**ire**, poliz**ia**, morte, eur**o**, genitor**i**, madr**e**, alcun**i**, local**i**, violenz**a**, arrest**ato**, pd;

15. **Topic 15**: enne, polizia, lavor**o**, morte, arrest**ato**, presidente, dichiar**ato**, vit**a**, bosc**hi**, pd, legg**e**, ospedale;

I tempi di esecuzione sono stati i seguenti:

- LDA con Spark: 2743 secondi;
- LSI: 7 secondi

L'intero processo di elaborazione è durato circa 46 minuti.

5.2 Conclusioni

Per quanto riguarda l'output, osserviamo diversi fenomeni relativi ai topic:

- *Primo cluster*: il primo cluster di parole ricopre un ruolo particolare, in quanto contiene i termini con il numero di occorrenze maggiore. Essa rappresenta una sorta di 'contenitore' delle parole-chiave dell'intero corpus. Al primo posto, per tutti, c'è la parola *stato*, probabilmente è l'ambiguità di significato di questo termine che gli conferisce il numero spropositato di occorrenze che possiede. Potrebbe essere utile eliminare tale parola durante una precedente procedura di stemming;

- *Ripetizione*: molte parole sono ripetute in diversi topic e, similmente, alcuni topic contigui possiedono quasi le stesse parole (es. *Repubblica*: Topic 3 e 4).

- *Sovrapposizione*: alcuni topic contengono due diversi macroargomenti fusi al loro interno (es. *La Stampa*: Topic 12 in cui sembra si fondano argomenti di politica estera e informazioni meteo d'attualità);

- *Interpretazione*: per alcuni topic è facile comprendere il tema di cui trattano (es. *Corriere*: Topic 11, in cui si parla di cronache sportive), per altri non sembra possibile (es. *Repubblica*: Topic 15). Alcuni topic accorpano parole che si rifanno a tematiche del tutto generali (es. *Fanpage*: Topic 3, 7, 8 in cui emerge il tema della violenza sulle donne), altri entrano addirittura nello specifico di alcune particolari vicende (es. Repubblica: Topic 9, in cui è palese la vicenda della visita del Papa in Myanmar).

Il fenomeno del *Primo Cluster* sono causati dalla riduzione che l'LSI opera. Ai primi posti dunque si collocano quei temi che spiegano una maggiore varianza (e che spesso hanno dunque un numero maggiore di occorenze). La *Sovrapposizione* e la similitudine di topic contigui può essere causata dal fatto che abbiamo prestabilito la lunghezza delle parole nel topic, oppure, nel primo caso, dal fatto che due argomenti interferiscono magari per mezzo di una parole di collegamento. L'interpretazione è generalmente possibile anche se ci possono essere casi in cui non si evince facilmente (o non è possibile inferire) il tema centrale.

Dalla nostra analisi non supervisionata possiamo trarre alcune conclusioni.

Innanzitutto si evince lo schema giornalistico delle 3 'S', e si vede che i temi principali trattati sono , *in primis*, politico-economici, e poi d'attualità e di cronaca (soprattutto nera e giudiziaria).

Repubblica e *Il Corriere*, testate di rilievo nazionale e le cui notizie raccolte sono un gran numero, trattano di argomenti principalmente di matrice

politico-economica. Non mancano, nella prima, vicende collegate all'attualità internazionale e all'istruzione, quest'ultimo tema presente anche nel *Corriere* che, inoltre, possiede un topic di cronaca sportiva che non si trova negli altri giornali.

Il Giornale, anch'esso di diffusione nazionale ma di cui abbiamo un minor numero di notizie a disposizione per l'analisi, è quasi interamente polarizzato verso la politica.

La Stampa, di cui abbiamo un numero di notizie simile a quello de *Il Giornale*, tratta di diverse tematiche, dalla politica all'attualità. Nel topic 8 si evince con sufficiente chiarezza la sua matrice territoriale.

Infine *Fanpage*, possiede caratteristiche che spiccano nel contesto dei giornali analizzati. Ultimo per fondazione e con diffusione esclusivamente on-line, i temi trattati sono diversi: il tema politico è sempre ai primi posti, ma non è quello più affrontato. Principalmente le vicende di attualità, soprattutto cronaca nera, si contendono il gran numero di topic. Fa eccezione il topic 4, l'unico in tutte le testate in cui si evince nettamente che il tema trattato sono le informazioni metereologiche.

Il fatto che alcuni temi non emergano nettamente nei topic esaminati, non vuol dire che i giornali non le trattino affatto. Probabilmente, con un numero maggiore di topic a disposizione e numerosi altri articoli in memoria, anche altre tematiche potrebbero emergere: tutto ciò che possiamo affermare è solamente che i temi principali degli articoli raccolti si rifanno a quelli estratti.

5.3 Commenti finali

Ci siamo preoccupati operare un'analisi non supervisionata dei contenuti, clusterizzando le tematiche principali con i topic model i file testuali appartenenti all'universo dei social media. Il risultato permette già di osservare l'emergere di alcuni schemi, come la regola giornalistica delle 3 'S', o le diverse polarizzazioni tematiche dei quotidiani che pubblicano su carta stampata e online. Abbiamo esaminato, inoltre, elementi tipici dell'output , elencando i fenomeni più comuni che si manifestano dalla procedura e cercando di fornire qualche spiegazione in merito.

Possiamo ora fare delle considerazioni conclusive e indicare possibili sviluppi.

Il caso di studio esaminato, costituisce un semplice esempio finale di applicazione del sistema distribuito, la cui implementazione, ribadiamo, è il vero tema centrale dell'elaborato. L'output è costituito, come abbiamo visto, da topic formati da parole estratte dai documenti.

A seconda del risultato che si desidera, si può procedere in diversi modi.

Si può voler definire un concetto di somiglianza tra quotidiani e verificare quali si assomigliano per tematiche trattate e quali, invece, sono dissimili; si

può volere, per ogni topic, assegnare un *label* che ne indichi sinteticamente il contenuto, oppure si possono associare i topic a diverse aree tematiche (nel caso del giornalismo possono essere Politica, Economia, Cronaca, ecc.), cioè operare un'analisi *supervisionata* dei topic.

Il concetto di analisi supervisionata mediante LDA è stato affrontato ed esistono degli algoritmi, evoluzioni dell'LDA e in particolare l'sLDA [Ble10], in grado di operare una tale classificazione.

Anche il concetto di *distanza*, tra due file testuali è stato sviluppato: mediante i topic estratti si viene a costituire una sorta di 'codice genetico' di ciascun documento e diversi documenti possono essere messi in relazione sulla base di un concetto appropriato di distanza[5] che misura le somiglianze tra i codici.

Un modo molto naive di assegnare un titolo sintetico a ciascun topic è quello di definire dei dizionari di parole-chiave (ad esempio si può associare *Economia* a parole come 'lavoro', 'euro', 'milioni', ecc.), così facendo si può capire quali sono i temi trattati da ciascun topic e si può operare una sorta di intuitiva analisi supervisionata (anche se la dicitura è impropria perché un topic potrebbe possedere parole che appartengono a diverse classi).

Ci siamo limitati a suggerire solamente queste possibili estensioni, tuttavia ciò consente di lasciare al nostro sistema una grande generalità di impiego nell'ambito dell'analisi dei social media, impiego che può essere poi specificato con modifiche esterne a seconda del particolare caso di studio.

Per quanto riguarda l'algoritmo in sé, possiamo suggerire, invece, diverse modifiche sulla base delle esigenze di studio.

Un aspetto importante è stato il fissaggio iniziale del numero di topic e del numero di parole di ciascun topic, per il quale ci siamo basati su prove empiriche. Tuttavia è possibile impostare la scelta e del numero di topic e delle parole che lo compongono sulla base di valori di soglia delle probabilità associate ai termini durante l'impiego dei topic model.

Un altro punto degno di nota è rappresentato dal testo specifico da esaminare. Nel nostro caso di studio, abbiamo fuso, per ciascun articolo raccolto, il suo titolo, il corpo del testo e i tags se presenti. Si potrebbe *pesare* ciascuna componente, in modo da dare rilievo, ad esempio, alle parole presenti in un titolo e nei tags raddoppiando le loro occorrenze: così facendo sarebbe più semplice per i topic model rilevare i termini chiave.

Infine, si potrebbe lanciare più volte l'LDA e fondere i risultati ottenuti: questo darebbe peso maggiore alle parole più rilevanti, ma avrebbe un notevole costo computazionale.

[5] Tipicamente per l'LDA viene utilizzata la *divergenza di Jensen-Shannon*, [Sch99]

Appendice A

Scraping da un social network

È generalmente più semplice reperire informazioni da un sito piuttosto che da un social network. Questo perché i social registrano informazioni molto personali la cui divulgazione potrebbe essere dannosa per la privacy dell'utente ed avere conseguenze infauste sia per il gestore del sito che per l'utente stesso. Pertanto lo scraping dei social network non avviene nelle stesse modalità descritte per i siti classici ed è necessario seguire una procedura differente allo scopo di tutelare la riservatezza di chi fruisce dei servizi del sito. In generale ciò vuol dire che, per gli esterni, non è sempre possibile avere accesso a tutta l'informazione contenuta nel social network che si sta analizzando, ma si può solo esaminarne una parte.

Scegliamo di analizzare la modalità di scraping un social network estremamente diffuso: Facebook.

La scelta è motivata dal fatto che Facebook, nel 2017, ha all'attivo 2 miliardi di utenti iscritti ed è il terzo sito più visitato al mondo [Ale17], per cui la quantità di informazione prodotta e analizzabile è veramente molta. Esso costituisce, inoltre, un buon rappresentante per le procedure di scraping, in quanto, con opportune modifiche, la seguente si può adattare anche ad altri social network come Twitter ed Instagram. In generale, l'ecosistema informatico dei social network e molto complesso ed è presunzione voler dare una visione completa in un'appendice, anche di uno solo dei social. La procedura presentata sarà una semplificazione, che ciononostante permetterà di reperire moltissime e utilissime informazioni da analizzare (ad esempio il contenuto dei post); per ulteriori approfondimenti si rimanda a [Rus13].

A.1 Autenticazione

Il primo passo per lo scraping di Facebook è la creazione di un account. Successivamente occorre effettuare l'autenticazione sul sito degli sviluppatori ([FacDev]) e la creazione di una nuova app durante la quale Facebook chiederà alcuni permessi. Si nota già la prima differenza con lo scraping da

siti classici: in questo caso lo scraping deve essere supervisionato dal sito su cui si sta eseguendo, le informazioni devono essere filtrate sulla base del livello di privacy e lo scraper deve essere riconosciuto per poter operare, anzi l'operazione di scraping viene vista come una vera e propria applicazione creata all'interno della piattaforma Facebook.

Ciò fatto occorre selezionare la voce *"Tool di esplorazione per la API Graph"*, nella cui schermata successiva viene fornito il token di accesso. Questo token è un codice alfanumerico di circa un centinaio di caratteri ed è la chiave di autenticazione dello scraper.

Per mettere in comunicazione l'ambiente Facebook con Python occorre, innanzitutto richiamare la libreria *facebook* tramite il comando **import** e successivamente inizializzare un connettore con il comando *facebook.GraphAPI*.

Il token di accesso non dura per sempre, il tempo a disposizione per la raccolta informazioni è di circa due ore. È chiaramente possibile richiederne un altro o aggiornare il precedente in automatico.

A.2 La struttura a grafo

L'ecosistema di Facebook è molto ramificato e in continua evoluzione. Nel nostro approccio semplificato, volto esclusivamente allo scraping di informazioni come commenti, post, like ecc. merita una trattazione più approfondita il sistema di incapsulamento delle informazioni che Facebook implementa. In particolare, per reperire le informazioni necessarie, l'impostazione del processo di querying è dettata dal Facebook Graph API. Quest'ultimo è un sistema intuitivo per la rilevazione dei dati che ci si può prefigurare come una struttura a grafo o a scatole cinesi, le cui informazioni vengono salvate in un comune file JSON. Specificando il percorso da seguire, come nelle comuni liste Python, si riesce a memorizzare l'informazione cercata. Generalmente i dati presenti comprendono sia informazioni che si palesano anche nel layout della pagina, sia dei *metadata* potenzialmente utili per le analisi.

Facebook è stata una delle prime società informatiche ad adottare l'Open Graph Protocol, il quale consente di estendere la struttura di grafo anche a siti esterni a Facebook, e dunque riuscire, supposto che un sito abbia accettato di includere le specifiche Facebook al suo interno, a reperire informazioni anche oltre l'universo proprio del social network.

A.3 Un esempio di codice e di output

Verifichiamo un esempio di codice e del modo di muoversi lungo il grafo. Preliminarmente impostiamo il token

```
import facebook
g = facebook.GraphAPI(access_token=token.string, version='2.7')
```

Prendiamo a modello la pagina social di *Repubblica*, supponendo di voler aggiungere all'analisi del capitolo 5 anche un'analisi dei post e dei commenti pubblicati sulla pagina ufficiale e di avere, dunque, necessità di caricare questi dati.

Occorre, innanzitutto, reperire il codice identificativo della pagina: per far ciò, ci sono siti specifici come [FinID].

Eseguiamo il seguente comando

```
pp(g.get_object(id = ID.Repubblica, fields = 'website, about,
engagement, category, fan_count, posts'))
```

Il cui risultato (parziale) è il seguente:

```
{
 'website': 'http://www.repubblica.it/',
 'about': 'Notizie, inchieste, approfondimenti,...',
 'engagement': {
  'count': 3540733,
  'social_sentence': '3.5M people like this.'
 },
 'category': 'Media/News Company',
 'fan_count': 3540733,
 'posts': {
  'data': [
   {
    'message': 'Le piccole hanno appena 7 mesi',
    'created_time': '2017-12-21T13:20:04+0000',
    'id': '179618821150_10156713588766151'
   },
   {
    'message': 'Ecco cosa ci faceva in Puglia',
    'created_time': '2017-12-21T13:00:23+0000',
    'id': '179618821150_10156713551096151'
   }, ...}
```

Con questo comando abbiamo chiesto che ci venissero mostrate informazioni riguardanti il sito, la sua descrizione, i post ecc., il formato è, come anticipato, di tipo JSON.

Per ciascun messaggio il grafo continua, tuttavia registrando il codice *id* di ciascuno, possiamo accedervi ed estrapolare i commenti con il comando

```
g.get_object(id = id_post, fields = 'message, comments')
```

il cui output è il seguente

```
{'comments':
{'data':
[{'created_time': '2017-12-21T07:25:09+0000',
'from': {'id': '10152833863517189', 'name': 'Al*****ro F***a'},
'id': '10156711589291151_10156711597426151',
'message': 'Io ancora non mi capacito, display che si autoriparano ...'},
{'created_time': '2017-12-21T10:05:18+0000',
'from': {'id': '10203015372527326', 'name': 'Ri****do C******o'},
'id': '10156711589291151_10156712295221151',
'message': 'Ma se inventasse modi per inquinare di meno ...'},
...}
```

per cui, individuando il percorso da seguire, ci accorgiamo che basta scrivere

```
g.get_object(id = id.post, fields = 'message, comments')['message']
```

per i post, e

```
grafo = g.get_object(id = id.post, fields = 'message,comments')
grafo['comments']['data']['message']
```

per i commenti.

Anche in questo caso occorre salvare il risultato ed impostare una procedura per la gestione dei duplicati. Valgono le medesime soluzioni viste in precedenza per MongoDB.

Appendice B

Richiami di teoria.

In questa appendice saranno richiamate le principali nozioni utili alla comprensione dei modelli illustrati nel capitolo 2.

B.1 Probabilità e statistica di base

Associamo ad un esperimento casuale un insieme Ω che definiamo *spazio campione*. Siamo interessati a definire su una collezione di elementi di Ω una misura della loro probabilità di occorrenza. Per rendere rigoroso il concetto di collezione di elementi di interesse dobbiamo riferirci alla seguente definizione:

Definizione 1. Una famiglia $\mathcal{F}$ di parti di Ω costituisce una σ-algebra su tale insieme, se soddisfa i seguenti requisiti:

1. $\Omega \in \mathcal{F}$;
2. se $A \in \mathcal{F} \Rightarrow \bar{A} \in \mathcal{F}$;
3. $\forall n \in \mathbb{N}, A_n \in \mathcal{F} \Rightarrow \bigcup_{n=1}^{\infty} A_n \in \mathcal{F}$.

Gli elementi di $\mathcal{F}$ saranno chiamati *eventi* e la coppia ordinata $(\Omega, \mathcal{F})$ si definisce spazio probabilizzabile. È sempre possibile, per ogni insieme Ω non vuoto, definire su di esso almeno una σ-algebra[1]. Si può sempre definire la più piccola σ-algebra (nel senso dell'inclusione insiemistica) che contiene fissati sottoinsiemi di Ω: tale σ-algebra si dice *generata* da questi eventi selezionati, che si definiscono a loro volta *generatori*.

Riveste particolare importanza la seguente σ-algebra:

Definizione 2. Sia $\Omega = \mathbb{R}$ e si scelga come insieme di generatori quello costituito dagli insiemi aperti di $\mathbb{R}$ secondo la topologia naturale: la σ-algebra da essi generata si chiama σ-algebra *di Borel* e i suoi elementi si dicono *Boreliani*.

[1] Si pensi ai casi estremi di $\mathcal{F} = \{\emptyset, \Omega\}$ o $\mathcal{F} = 2^{\Omega}$.

Diamo la seguente, cruciale, definizione:

Definizione 3. Sia assegnato uno spazio probabilizzabile $(\Omega, \mathcal{F})$. Una funzione $\mathbb{P} : \mathcal{F} \to \mathbb{R}$ si chiama *misura di probabilità* su $(\Omega, \mathcal{F})$ se soddisfa le seguenti proprietà:

1. $\forall A \in \mathcal{F}, \mathbb{P}(A) \geq 0$;
2. $\mathbb{P}(\Omega) = 1$;
3. (Numerabile Additività) per ogni successione $\{A_n\}_{n\in\mathbb{B}}$ di eventi incompatibili[2] si ha
$$\mathbb{P}\left(\bigcup_{n=1}^{\infty} A_n\right) = \sum_{n=1}^{\infty} \mathbb{P}(A_n).$$

La terna $(\Omega, \mathcal{F}, \mathbb{P})$ si definisce *spazio di probabilità*.

In generale possiamo avere interesse a considerare la probabilità di un evento *condizionatamente* ad un altro. Vale la seguente definizione

Definizione 4. Sia fissato $(\Omega, \mathcal{F}, \mathbb{P})$. Sia dato, inoltre, $B \in \mathcal{F}$ un evento a probabilità non nulla e $A \in \mathcal{F}$ un evento fissato. La seguente espressione

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$$

si chiama *probabilità di A condizionata a B* e si verifica essere un'altra misura di probabilità su $(\Omega, \mathcal{F})$.

Se vale $\mathbb{P}(A|B) = \mathbb{P}(A)$, gli eventi A e B si dicono *indipendenti*.

Con questa definizione possiamo enunciare, senza dimostrazione, il teorema di Bayes

Teorema 1. Sia $\{H_1, H_2, \cdots, H_k\}$ una partizione di Ω e sia B un evento a probabilità non nulla. Vale la seguente relazione

$$\mathbb{P}(H_i|B) = \frac{\mathbb{P}(H_i)\mathbb{P}(B|H_i)}{\sum_{j=1}^{k} \mathbb{P}(H_j)\mathbb{P}(B|H_j)}$$

Il Teorema di Bayes, generalmente, si interpreta come un meccanismo si aggiornamento della conoscenza circa un evento, dato che un altro evento B si è verificato. Il vettore delle probabilità $(\mathbb{P}(H_1), \mathbb{P}(H_2), \cdots, \mathbb{P}(H_k))$ è il chiamato vettore delle probabilità a *iniziali* o a *priori*, mentre il vettore $(\mathbb{P}(H_1|B), \mathbb{P}(H_2|B), \cdots, \mathbb{P}(H_k|B))$ contiene le probabilità a *posteriori* o *finali*. Ci concentriamo ora sul concetto fondamentale di *variabile aleatoria*. Diamo la seguente definizione

[2] $A_i \cap A_j = \emptyset$ per ogni $i \neq j$.

Definizione 5. Sia H un insieme, $\mathcal{H}$ una σ-algebra su H e g un'applicazione di H in $\mathbb{R}$. Se la controimmagine mediante g di ogni insieme di Borel di $\mathbb{R}$ è un elemento di $\mathcal{H}$, allora si dice che g è $\mathcal{H}$-misurabile.

L'appellativo di *variabile* può trarre in inganno, in quanto la variabile aleatoria è in realtà una *funzione*, come mostra la seguente definizione:

Definizione 6. Dato uno spazio di probabilità $(\Omega, \mathcal{F}, \mathbb{P})$, una funzione $X : \Omega \Rightarrow \mathbb{R}$ si chiama *variabile aleatoria* se essa è $\mathcal{F}$-misurabile.

Supponendo fissato lo spazio di probabilità $(\Omega, \mathcal{F}, \mathbb{P})$ con associato una variabile aleatoria X definiamo:

Definizione 7. Si chiama *funzione di distribuzione* della variabile aleatoria X l'applicazione $F_X : \mathbb{R} \to [0, 1]$ definita come segue:

$$\forall x \in \mathbb{R}, F(x) := \mathbb{P}(]-\infty, x]) \equiv \mathbb{P}(X \leq x)$$

Esaminiamo le due funzioni di distribuzione principali a cui ci riferiamo nei modelli: la distribuzione Multinomiale e quella di Dirichlet. La prima è una distribuzione discreta, la seconda è assolutamente continua ed entrambi sono distribuzioni multiple. Definiamo innanzitutto questi ultimi concetti prima di scrivere esplicitamente le distribuzioni.

Definizione 8. Sia X una variabile aleatoria e $F(x)$ la sua funzione di distribuzione. Si dice che X è una variabile aleatoria discreta se il suo supporto è finito o numerabile e se esiste una funzione $p : X \Rightarrow [0, 1]$, detta *massa* di probabilità, tale che

$$F(\bar{x}) = \sum_{x \leq \bar{x}} p(x)$$

in particolare, la funzione di ripartizione si presenta come una funzione costante a tratti.

Vale la seguente

Definizione 9. Sia X una variabile aleatoria e $F(x)$ la sua funzione di distribuzione. Si dice che X è una variabile aleatoria assolutamente continua e lo stesso si dice di $F(x)$, se esiste una funzione non negativa f, detta *densità*, tale che:

$$F(x) = \int_{-\infty}^{x} f(t)dt \qquad \forall x \in \mathbb{R}$$

Le definizioni di variabile aleatoria e funzione di ripartizione, sono state date nel caso monodimensionale che è il più semplice e il meno generale. Tali definizioni, però, si estendono con facilità anche a più dimensioni e si ottengono così le variabili aleatorie multiple. Intuitivamente, le variabili aleatorie avranno come codominio $\mathbb{R}^n$, Non entreremo nei dettagli di questa

estensione e delle problematiche che solleva rispetto al caso di dimensione singola. Per approfondimenti rimandiamo a [Dal03].

Una variabile aleatoria X che si distribuisca come una Multinomiale, si scrive $X \sim Mu(\nu, \theta_1, \theta_2, \cdots, \theta_k)$ ed ha funzione di massa pari a

$$p(x_1, x_2, \cdots, x_k) = \frac{\nu!}{x_1!, x_2! \cdots x_n!} \theta_1^{x_1} \theta_2^{x_2} \cdots \theta_k^{x_k}$$

dove $x_i = 0, 1, \cdots, \nu$, $\sum x_i = \nu$ e $\sum \theta_i = 1$ con $\theta_i > 0$ per ogni indice.

La densità della distribuzione di Dirichlet è la seguente:

$$f(x_1, x_2, \cdots, x_k) = \frac{\sum_{i=1}^{k} a_i}{\prod_{i=1}^{k} \Gamma(a_k)} x_1^{a_1 - 1} x_2^{a_2 - 1} \cdots x_k^{a_k - 1}$$

dove ciascun x_i è non-negativo, $\sum_{i=1}^{k} x_i = 1$ e ciascun a_i è strettamente positivo. Tale formula va riguardata come una densità con supporto $k-1$-dimensionale per via della restrizione alla somma delle x_i.

Rammentiamo che la funzione *Gamma* ha la seguente forma:

$$\Gamma(x) = \int_0^\infty t^{x-1} e^t dt, \text{ con } x > 0.$$

Sia $S \subset \mathbb{R}^m$ con $m \geq 1$; una classe $\{f_\theta, \theta \in I\}$ di funzioni di densità indicizzate da un parametro e con supporto S si dice essere una *famiglia esponenziale* se si può scrivere:

$$f_\theta(\mathbf{x}) = h(x) \exp\{\eta(\theta) T(\mathbf{x}) - B(\theta)\} \qquad \mathbf{x} \in S$$

per una opportuna scelta delle funzioni A, B, h, η.

Si verifica facilmente che la distribuzione di Dirichlet è di tipo esponenziale, ponendo:

$$h(x) = 1$$

$$\eta(\theta) = \alpha - 1$$

$$T(\mathbf{x}) = \log \mathbf{x}$$

$$B(\theta) = \left(\sum_k \log \Gamma(\theta_k) - \log \Gamma \left(\sum_k \theta_k \right) \right).$$

B.2 Statistica di base

È centrale la nozione di esperimento statistico, definito dalla seguente

Definizione 10. Un esperimento statistico è una famiglia di spazi di probabilità $e = \{(\Omega, \mathcal{F}, \mathbb{P}_\theta), \theta \in I\}$

L'insieme I rappresenta lo spazio delle ipotesi: in esso si suppone di trovare il 'vero' valore di θ, ovvero quello che meglio è in grado di modellizzare l'esperimento. La ricerca di θ avviene sulla base di realizzazioni dell'esperimento, ciascuna delle quali si indica con ω è appartiene all'insieme Ω.

Dato un esperimento statistico assegnato, una qualsiasi applicazione misurabile $T : \Omega \to \mathcal{T}$ se $\mathcal{T}$ è misurabile, si dice *statistica*. Ogni statistica può essere riguardata come una variabile aleatoria e dunque possiederà una propria distribuzione di probabilità, detta in questo caso *campionaria*.

Il valore assunto da una statistica, che è utilizzata per la ricerca dell'ipotesi vera (detta procedura di *inferenza* puntuale su θ), si chiama *stima puntuale* e la statistica in questione viene detta *stimatore puntuale*.

Per le analisi dell'esperimento, inoltre, la funzione di verosimiglianza rappresenta uno strumento fondamentale per lo studio dei risultati statistici; ne diamo una definizione nel caso discreto, ma è facile estenderla al caso continuo.

Definizione 11. Sia dato un esperimento statistico $e = \{(\Omega, \mathcal{F}, \mathbb{P}_\theta), \theta \in I\}$ tale che $\mathbb{P}_\theta$ sia discreta per ogni θ, si chiama funzione di *verosimiglianza* associata al risultato $\omega_0 \in \Omega$ la funzione $l : I \Rightarrow [0, 1]$ definita da:

$$l : \theta \to \mathbb{P}_\theta(\omega_0)$$

La verosimiglianza di un'ipotesi è dunque la probabilità che si assegnerebbe a priori al risultato ω_0 se θ fosse assunta come ipotesi vera.

Essa possiede diverse proprietà, la più importante delle quali è che al crescere della dimensione del campione ω, la funzione di verosimiglianza converge in probabilità al valore vero dell'ipotesi.

Diamo adesso una definizione di statistica sufficiente

Definizione 12. Dato un esperimento statistico $e = \{(\Omega, \mathcal{F}, \mathbb{P}_\theta), \theta \in I\}$, si dice che la statistica $T : \Omega \to \mathcal{T}$ è *sufficiente* se la funzione di verosimiglianza si decompone nel seguente modo:

$$l(\theta; \omega) = \gamma(\omega) \cdot \varphi(\theta, T(\omega)) \forall (\theta, \omega) \in I \times \Omega$$

dove $\gamma : \Omega \Rightarrow \mathbb{R}$ e $\varphi : I \times \mathcal{T} \to \mathbb{R}$

La statistica sufficiente, dunque, concentra in essa una sintesi dell'intera informazione dell'esperimento circa l'ipotesi da determinare.

Il teorema di Pitman-Koopman-Darmois [Koo36], asserisce che le famiglie esponenziali sono le sole per cui le statistiche sufficienti restano limitate anche se la numerosità del campione tende all'infinito. Questa è una proprietà anche della distribuzione di Dirichlet in quanto abbiamo visto in precedenza che essa appartiene alla famiglia esponenziale.

Resta un ultimo concetto da chiarire: quello di modello bayesiano. Tale aggettivo viene usato riguardo ad ogni metodo di analisi statistica in cui si presuppone un'assegnazione di probabilità a ogni evento incerto. Nel caso dell'esperimento statistico, lo spazio a cui ci si riferisce da un punto di vista del modello di impostazione bayesiana, è il seguente: $(I \times \Omega, \mathcal{F}_{I\times\Omega}, \mathbb{P}_{I\times\Omega})$, ossia anche lo spazio delle ipotesi è probabilizzato. In risposta ad ogni problema inferenziale, nel modello bayesiano, si otterrà una legge di probabilità. Lo schema operativo è dettato dal teorema 1, che riproponiamo adesso in versione continua e con una notazione leggermente differente ma più congeniale alla spiegazione che ne daremo in seguito.

Lo schema è il seguente:

$$\pi(\theta;\omega) = \frac{\pi(\theta) f_\theta(\omega)}{\int_I \pi(\theta) f_\theta(\omega) d\theta}$$

dove f_θ è la densità associata a $\mathbb{P}_\theta$, $\pi(\cdot)$ viene definita legge di probabilità a *priori* e $\pi(\cdot;\omega)$ e la legge di probabilità a *posteriori* di Θ condizionata a $\Omega = \omega$.

Una tale impostazione ha carattere dinamico: assegnata una probabilità a priori, da questa si ottiene una a posteriori, la quale può essere reinserita di nuovo nel meccanismo come una densità a priori dato un nuovo esperimento e così via, aggiornando continuamente al crescere dell'informazione.

In quest'ottica, la seguente definizione (per ulteriori dettagli [Pic09]) riveste particolare importanza

Definizione 13. Dato il modello $e = \{(\Omega, \mathcal{F}, \mathbb{P}_\theta), \theta \in I\}$, una classe di distribuzione su I si dice *coniugata* al modello se, prendendo in tale classe la distribuzione iniziale, anche la distribuzione iniziale, qualunque sia $\omega \in \Omega$, vi appartiene.

In conclusione, mostriamo come la distribuzione Multinomiale , sia coniugata a quella di Dirichlet. Ciò si vede facilmente in quanto, abbiamo che, scegliendo $\pi(\theta) \sim Dir(\theta|\alpha)$ e $f_\theta \sim Mu(\nu, \theta)$, a meno di un fattore indipendente da θ si ha:

$$Mu(\nu,\theta)Dir(\theta) \sim \prod_{k=1}^{m} \theta_k^{x_k} \prod_{k=1}^{m} \theta_k^{\alpha_k - 1}$$

$$\sim \prod_{k=1}^{m} \theta_k^{x_k + \alpha_k - 1} = Dir(x + \alpha)$$

per cui è verificato il coniugio

Bibliografia e Sitografia

[Ale17] Alexa, *The top 500 sites on the web* https://www.alexa.com/topsites/global;0, 2017

[BioEN] Biografie di Albert Einstein ed Isaac Newton, https://it.wikipedia.org/wiki/Albert_Einstein; https://it.wikipedia.org/wiki/Isaac_Newton

[BeaSp] BeautifulSoup libreria Python, https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[Ble03] Blei David M., Ng Andrew Y., Jordan Michael I. *Latent Dirichlet Allocation*, Journal of Machine Learning Research, pp. 993-1022, 2003.

[Ble10] Blei David M., J. D. McAuliffe *Supervised topic models*, Arxiv, 2010.
https://papers.nips.cc/paper/3328-supervised-topic-models.pdf

[Con12] P.L. Conti, D. Marella, *Campionamento da popolazioni finite*, Springer, 2012

[CorRSS] Indirizzo Feed RSS de *Il Corriere della Sera*, http://xml.corriereobjects.it/rss/homepage.xml

[Dal03] G. Dall'Aglio , *Calcolo delle probabilità*, Zanichelli, 1999.

[Dar11] W. M. Darling , *A Theoretical and Practical Implementation.Tutorial on Topic Modeling and Gibbs Sampling*, University of Guelph, 2011.

[DeFin] B. de Finetti, *De Finetti's Theorem*, https://www.encyclopediaofmath.org/index.php?title=De_Finetti_theorem, Encyclopedia of Mathematics.

[Der90] Deerwester S., Dumais, S. T., Furnas, G. W., Landauer, T. K., Harshman, R., *Indexing by latent semantic analysis*, Journal of the American Society for Information Science 41(6), 391-407, 1990.

[FacDev] Facebook for Developers, https://developers.facebook.com/

[FanRSS] Indirizzo Feed RSS di *Fanpage*, https://www.fanpage.it/feed/

[FinID] Esempio di servizio di reperimento ID di Facebook, *Find my Facebook id* https://findmyfbid.com/

[GooN] Google News, https://news.google.com/news/

[Hof99] Hofmann, T. , *Probabilistic latent semantic indexing*, Proceedings of the Twenty-Second Annual International SIGIR Conference, 1999.

[IlGRSS] Indirizzo Feed RSS de *Il Giornale*, http://www.ilgiornale.it/feed.xml

[IntLS] Internet Live Stats, *Twitter Usage Statistics*, http://www.internetlivestats.com/twitter-statistics/

[Koo36] B. O. Koopman, *On Distribution admitting a sufficient statistic*, American Mathematical Society, 1936.

[LaSRSS] Indirizzo Feed RSS di *La Stampa*, http://www.lastampa.it/rss.xml

[Lin16] Lin Liu, Lin Tang, Wen Dong, Shaowen Yao and Wei Zhoucorresponding, *An overview of topic modeling and its current applications in bioinformatics* https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5028368/, SpringerOpen, 2016

[Lym03] P. Lyman, H. R. Varian, K. Swearingen, P. Charles, N. Good, L.L. Jordan, and J. Pal, *How much information?*, http://www2.sims.berkeley.edu/research/projects/how-much-info-2003, 2003

[NaLTK] nltk, documentazione della libreria Python, http://www.nltk.org/

[MonDB] MongoDB, https://www.mongodb.com/it

[Pic09] L. Piccinato, *Metodi per le decisioni statistiche*, Springer, 2009.

[PyMDB] PyMongo, documentazione della libreria Python, https://api.mongodb.com/python/current/

[PySpk] Pyspark, documentazione della libreria Python, http://spark.apache.org/docs/latest/api/python/

[QuoSfH] Quora, risposta alla domanda 'Is Apache Spark faster than Hadoop processing?', https://www.quora.com/Is-Apache-Spark-faster-than-Hadoop-processing

[QuoSvP] Quora, risposta alla domanda 'What is the difference between using Spark in Scala and Python?', https://www.quora.com/What-is-the-difference-between-using-Spark-in-Scala-and-Python

[Rad17] The Radicati Group Inc. *Email Statistics Report, 2017-2021*, http://www.radicati.com/wp/wp-content/uploads/2017/01/Email-Statistics-Report-2017-2021-Executive-Summary.pdf, 2017

[RepRSS] Indirizzo Feed RSS di *Repubblica*, http://www.repubblica.it/rss/homepage/rss2.0.xml

[Roe12] C. Roe, *The growth of unstructured data: what to do with all those zettabytes?*, http://www.dataversity.net/the-growth-of-unstructured-data-what-are-we-going-to-do-with-all-those-zettabytes/, 2012

[Rus13] M. Russell, *Mining the Social Web, 2nd Edition*, O'Reilly Media Research, 2013.

[San15] Sandy Ryza, Uri Laserson, Josh Wills, Sean Owen *Advanced Analytics with Spark*, O'Reilly Media Research, 2015.

[San17] Vicenda sottomarino *San Juan*, https://it.wikipedia.org/wiki/ARA_San_Juan_(S-42), 2017.

[Sal88] G. Salton, *Term-weighting approaches in automatic text retrieval*, Inform. Process. Man. 24(5), 513-523. , 1988

[Sch99] H. Schütze, C. D. Manning *Foundations of Statistical Natural Language Processing*, MIT Press, 1999.

[Scrpy] Scrapy, https://scrapy.org/

[SckLn] Scikit-Learn, Libreria Python, http://scikit-learn.org/stable/

[Zha16] C.X. Zhai, S. Massung *Text Data Management and Analysis: A Practical Introduction to Information Retrieval and Text Mining* , Acm Books, 2016.

Ringraziamenti

Ringrazio il mio Relatore, il Professor Umberto Ferraro Petrillo, per avermi consentito di confrontarmi con un problema di ampio respiro, molto attuale e per mezzo del quale ho imparato molto.

Parimenti ringrazio il mio Correlatore, il Professor Agostino di Ciaccio, per i suoi preziosi suggerimenti che hanno portato, in particolare, alla fusione delle due metodologie di topic model.

Rinnovo i ringraziamenti, espressi nella precedente tesi, a tutte le persone che mi sono state vicino anche in questo percorso. Ad esse se ne aggiungono di nuove.

Ringrazio di cuore i miei i colleghi, in particolare Francesco Curia e Michele Cianfriglia, per avermi dato preziosi consigli durante questi anni, ma anche per i bei momenti insieme.

Ringrazio mio padre perché, per mia fortuna, non teme di alzarsi presto la mattina...

Alla persona cui è dedicato l'intero lavoro va il mio ringraziamento più grande: evito di spiegarne i motivi, anche perché la lista sarebbe lunga almeno il doppio di questa tesi.