Indice

Introduzione

Questo lavoro di tesi si propone lo scopo di progettare ed implementare un programma in grado di acquisire, processare ed estrarre i contenuti 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 dei concetti chiave in modo da clusterizzare un certo numero di documenti secondo l'argomento trattato; 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 da poter essere generalizzato e suscettibile di future applicazioni.

A culmine di tutta l'opera 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 con l'obiettivo 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 una verità apodittica.

Nel Capitolo 2 vengono presentati i principali topic model: 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 3 si presentano uno per uno i software utilizzati per 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 dell'algoritmo a livello distribuito.

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 di clusterizzazione interamente non supervisionata.

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

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 nel layout per facilitarne l'impaginazione.

Capitolo 1

Il problema trattato

Secondo uno studio dell'Università di Berkeley [], 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 "; inoltre, un recente reportage [] stimava che dal 2005 al 2020 la produzione totale di informazione presente in rete sarebbe cresciuta di un fattore di scala pari a 300, previsione che, ad oggi, si è rivelata corretta [].

Queste informazioni sollevano diverse riflessioni. Si impone la necessità di gestire un tale flusso di dati, che, dato l'enorme volume che li caratterizza, sono stati definiti dalla letteratura con l'aggettivo *big*: i *big data* rappresentano un punto di svolta nell'analisi dei dati e, in una certa misura, una sfida sia per la loro gestione e sia per la loro interpretazione.

1.1 Problematiche legate ai big data

È facile intuire quali siano le problematiche legate ai big data. Riconoscendo che, da un lato, un tale flusso di informazione non può essere ignorato e, dall'altro, per un essere umano singolo è impensabile portare a termine un analisi efficiente in tempi accettabili, si comprende la necessità del sostegno informatico in questo compito.

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

[1] 1 Terabyte = 1000 Gigabyte

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, se non processati correttamente, portano a conclusioni inevitabilmente distorte []. Di questa importante tematica non ci occuperemo in questa sede e l'abbiamo enunciata solamente per completezza. Per approfondimenti si rimanda a [][].

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 naturali funzioni: occorre istruirla alla comprensione del *linguaggio naturale*. 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 della probabilità e della statistica, la macchina riesce a riconoscere i termini di maggiore rilevanza, i costrutti latenti e concetti più complessi come la sinonimia.

Per comprendere come si possa istruire un computer a far ciò, occorre fare uno sforzo e immaginare in che modo sia possibile comprendere un testo in al modo delle macchine, avulso, cioè, dall'uso delle facoltà di interpretazione propriamente umane. Il modo più facile di figurarsi ciò 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 si intravedono: 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 in seguito. È importante, invece, capire come un siffatto approccio si può 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. 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, sicché i topic rappresentano l'archetipo generativo della forma finale del testo. 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 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*.

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

1.3 La scelta del caso di studio

I contenuti testuali presenti in rete sono moltissimi e disparati e vanno dalle mail ai tweet, dai blog ai siti istituzionali. In particolare, la nostra scelta è ricaduta sul settore dell'editoria giornalistica i cui contenuti vengono messi a disposizione in rete: essi rappresentano una parte molto rilevante della categoria più generale dei *social media*, dove con questa locuzione si intende estendere alla rete la realtà una volta esclusiva dei *mass media*. In questo senso il giornalismo rappresenta una forma di contatto tra queste due dimensioni, tuttavia le motivazioni della nostra scelta sono diverse e molteplici. Innanzitutto, il giornalismo produce articoli ad un ritmo costante, il che è fondamentale per le rilevazioni: nel caso di tweet o mail, i contenuti possono essere postati in modo occasionale e dunque non è scontato che si riesca ad accumulare un sufficiente numero di dati in tempo fissato. 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 Facebook chiudano spontaneamente, mentre è rarissimo che un giornale di punto in bianco smetta di esistere. Per di più, le conclusioni che possiamo trarre dall'analisi degli articoli di gior-

nale hanno 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, tuttavia nulla vieta che la procedura sviluppata possa essere riutilizzata per altri scopi. Sebbene non ci spingeremo oltre i nostri propositi, possiamo suggerire qualche scenario, sulla base anche di alcune idee [][]

Capitolo 2

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. Un algoritmo in grado di effettuare una tale operazione viene definito *topic model*. Ci concentreremo principalmente su due topic model, il Latent Semantic Indexing ed il Latent Dirichlet Allocation (di seguito abbreviati LSI e LDA) focalizzandoci, per quanto concerne il case study, principalmente sul secondo.

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. []. Tale metodologia è stata adottata in modo pressoché incontrastato fino alla fine del secolo scorso nell'ambito dei motori di ricerca []. Successivamente, Hoffman [] nel 1999, ha generalizzato la procedura implementando il *Probabilistic Latent Semantic Indexing* (pLSI) di cui daremo qualche cenno. Infine, nel 2003, Blei et al. [] hanno realizzato l'algoritmo di LDA, che ha raggiunto grande fama e diffusione.

2.1 Latent Semantic Indexing.

Il *Latent Semantic Indexing*, spesso chiamato anche *Latent Semantic Analysis*, segue un approccio *bag of words* e consiste in una riduzione dimensionale di una matrice parole-documenti che vedremo in dettaglio.

2.1.1 Schema *tfidf*.

Definiamo la seguente matrice

$$DTM^1 = \{w_{ij}\}_{i=1,2,\cdots,T;j=1,2,\cdots,D}$$

[1] Sta per *Document Term Matrix*

ove T rappresenta il numero totale dei termini presenti in tutti i documenti e D il numero di tali documenti. Pertanto l'elemento w_{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] nella forma:

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

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

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

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 non riesce a rilevare aspetti del linguaggio naturale, anche se riesce ad enucleare i termini maggiormente esplicativi nei documenti.

2.1.2 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

[2]Che sta per term frequency-inverse document frequency

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.

2.1.3 Un esempio di applicazione

2.1.4 Cenni al *pLSI*

Pur senza addentrarci nei dettagli, enucleiamo i punti principali 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 medello 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 e, per far ciò, si considera un'ulteriore variabile, poniamo z, 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.

2.2 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.

2.2.1 Definizioni e formalizzazione.

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

Sebbene in precedenza abbiamo omesso definizioni di concetti quali parola, documento e corpus, lasciando all'intuito il compito di chiarirli, nel prosieguo, con l'obiettivo di una maggiore profondità e di un maggiore dettaglio, è necessario fornire le seguenti definizioni:

1. Un dizionario è un insieme finito del tipo $\{1, \ldots, V\}$, 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 D $=\{w_1, w_2, \ldots, w_M\}$

Passiamo ora alla 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:

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

In cui β è una matrice di topic tale per cui $\beta_{kj} = p(w^j = 1|z = k)$ 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:

$$p(\theta|\alpha) = \frac{\Gamma(\sum_{k=1}^{K} \alpha_k)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \dot{\theta}_1^{a_1-1} \dot{\theta}_2^{a_2-1} \cdots \dot{\theta}_K^{a_K-1}.$$

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.

La seguente figura riassume quanto esposto.

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

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

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

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

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.

2.2.2 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

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

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 Expevtation Maximization); il lower bound è costituito da un'approssimazione della distribuzione a posteriori sulle variabili latenti che appartiene alla seguente famiglia

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

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

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

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.

2.2.3 Considerazioni e confronti.

Capitolo 3

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.

3.1 Database NoSQL.

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*[].

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.

3.1.1 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.

3.1.2 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.

3.2 Sistemi di gestione distribuiti e Apache Spark.

La scelta di utilizzare Apache 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 sia noto [] che 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.

La scelta di operare con linguaggio Python (che motiveremo in seguito), rallenta notevolmente [] 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 fino dieci volte superiori.

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, per approfondimenti segnaliamo [].

3.2.1 Struttura e funzionamento di Spark.

Essenzialmente Spark è un sistema di computazione cluster distribuito e ad alte prestazioni. In figura è esemplificata la struttura delle sue componenti e distribuzione della gestione dei processi.

In particolare il *Cluster Manager* centrale gestisce il *Driver Program* e i diversi *Worker Nodes*. In Particolare ciascun *Worker Node* esegue diverse *Tasks*, cioè compiti, tra i quali l'avvio dei DAG (Direct Acyclic Graphs).

3.2.2 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 trasformazione, di un filtro o un'operazione di mapping, 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.

3.3 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.

3.3.1 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* [], 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*, 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.

3.3.2 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*[]. 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.

3.3.3 *Natural Language Toolkit*

Il *Natural Language ToolKit* (di seguito *nltk*) è un insieme di librerie 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, rendendo il testo uniformemente 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.

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. In genere 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, inoltre 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 informazione: la prima affermazione è certamente vera, sulla seconda bisogna essere cauti. 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 stop remover, 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 perdite; tuttavia occorre sempre tenere in conto che non è mai possibile ridurre il contenuto di un documento senza che ciò comporti una 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 (...). 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 nè 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').

Sebbene in questo caso la perdita di informazione sia necessariamente più marcata, questa procedura aumenta generalmente le prestazioni dell'analisi.

3.3.4 Scikit-Learn e pypark

Il cuore dell'analisi è costituito dalle procedure di analisi testuale che si applicano ai dati memorizzati. Per quanto riguarda l'LSI, l'algortimo si basa su funzionalità built-in della suite Scikit-Learn, che consiste in una libreria di procedure per il Machine Learning []. Le funzioni principali che di essa ci riguardano sono:

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).

Il tutto è implementato senza l'ausilio di Spark, per ragioni che saranno chiarite nei due capitoli finali. Viceversa la procedura LSA è interamente eseguita all'interno dell'ambiente Spark il quale viene richiamato mediante pyspark, ovvero una libreria API che consente di richiamarne la shell. In Spark è implementato l'algoritmo che esegue la procedura LDA

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 moduli, ognuno dei quali assolve a una funzione.

4.1 Raccolta e storage dei dati

La raccolta dei file di testo avviene con Scrapy, il quale è impostato per funzionare dal prompt dei comandi i cui comandi richiamano, dalla cartella del progetto, i file contenenti diversi sottomoduli, ciascuno dei quali va preimpostato indicando le specifiche del sito e 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 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 ; 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*, il quale rappresenta il cuore del nostro scraper. In esso si specifica l'URL del sito da analizzare e il codice HTML che consente di reperire 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 accedere al loro contenuto è necessario reperire l'URL i ciascuna notizia. Essa è chiaramente presente nel codice della pagina iniziale (in quanto cliccando su una specifica notizia siamo trasportati alla pagina che la riguarda) e dunque basta reperire l'indirizzo specifico 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.

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

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.

A questo punto i file sono salvati in una collection la quale può essere estratta è inserita in un file di testo. La particolarità del file così ottenuti è quella di essere composti da diverse righe di testo in formato JSON. Spendiamo qualche parola in più su questo formato che ha una certa importanza per via della sua leggerezza e della sua flessibilità di utilizzo.

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: ini 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 identiche al computer 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, ciascuno dei quali può in qualche misura migliorare la comprensione del testo per la macchina.

La procedura di rimozione delle stopwords è contenuta nella routine ***clean***. Il passo iniziale è quello di richiamare il dizionario delle stopwords di *nltk*

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

ad esso vanno aggiunte alcune altre parole di uso comune che il dizionario impostato non comprende. Il codice di esempio è il seguente

Nella routine, per prima cosa vanno eliminate alcune 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, e si escludono numeri, segni di punteggiatura e tutte le lettere singole. Il testo ottenuto viene ridotto tutto a caratteri minuscoli.

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

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

Effettuata la procedura, il risultato è simile a questo

```
utilizz dat minister istruzion lanc allarm poch
prof matemat quind cattedr ruol rimang vuot problem
avvert soprattutt scuol med quest anno cre voragin
post rimast vacant dop trasfer soprattutt nord lombard
sol arriv par nemmen prossim immission ruol colm...
```

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

Onde evitare di ottenere un output dalla procedura formato da parole di cui non se ne riconosce il significato, è stata creata un altra routine di nome ***rebuild*** la quale associa una radice semantica a 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.

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; nell'algoritmo esso è stato implementato all'interno dell'ambiente *Spark Context* richiamato dalla libreria *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 di parole fissato per ciascun concetto a priori prima di lanciare il programma. 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 che topic con 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. Il risultato dell'analisi LDA sarà simile al seguente (caso $n = 10$ e $k = 2$):

```
Topic 1:
matem  0.009560573221092942
scuol  0.00808030602233562
post  0.007174175578550224
vuot  0.00701157776472776
quas  0.006984416666260308
capac  0.006773406498557413
```

```
istr  0.0066294229384543345
ben  0.006600503209852239
aver  0.006556311017571411
reclut  0.0065142697691635085


Topic 2:
frut  0.00755124839871547
vorag  0.007512074134983625
lombar  0.007231254609200753
profession  0.007015777869361581
fedel  0.006789730066915294
dat  0.006683804983393134
minis  0.006668940226489626
confr  0.0066030669225653794
fa  0.00660271421692034
par  0.006571353930747912
```

Come si vede ciascuna parola è affiancata da una probabilità che è tanto più alta quanto più, secondo il modello, quella parola ha affinità con il topic.

Poiché l'obiettivo finale è quello di una classificazione di un intero corpus di argomenti secondo i temi in essa trattati, dobbiamo immaginare che diversi output del genere vengano memorizzati, ciascuno per ogni documento che viene sottoposto all'LDA. Il file completo di tutti questi output 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 cluser di parole, ciascuno dei quali può simboleggiare un concetto importante nella collezione. La provenienza di questi termini è ciascun topic che proviene dall'LDA. Si nota, nell'output appena mostrato, che ci sono parole come 'quas' (che sta per 'quasi') nel primo topic, o 'fa' (dal verbo 'fare') nel secondo. 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'output sarà simile al seguente:

```
...
```

...
...
...

il quale può essere reso intelligibile grazie alla routine descritta sopra. Anche in questo caso abbiamo delle probabilità accanto alle parole, 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 sono effettivamente cariche di significato per l'LDA, in un cluster 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* (nel nostro caso è stato riadattato un dizionario già in possesso).

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 necessario utilizzare il modello di tipo probabilistico, per le ragioni concernenti i limiti di un modello deterministico 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 importanti per una classificazione di un livello più alto; 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 questi due algoritmi che, non è escluso, possono fornire risultati degni di nota. Tuttavia nel caso di un'applicazione a due stadi dell'LSI mancherebbe completamente la dimensione probabilistica, ovvero non si dà 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 qua-

li motivi non è consigliato scambiare le due metodologie in questa analisi, ovvero eseguire prima l'LSI e poi l'LDA.

Capitolo 5

Caso di Studio: Risultati

Appendice A

Scraping da un social network

È generalmente più semplice reperire informazioni da un sito piuttosto che da un social network. Questo perché generalmente 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 [], 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 avere la presenza di riuscire a dare una visione completa del tutto 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 [][].

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 [] 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 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 [].

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.

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 estremali 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 [].

Una variabile aleatoria X che si distribuisca come una Multinomiale, si scrive $X \sim Mu(\ni, \theta_1, \theta_2, \cdots, \theta_k)$ 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 $x_i = 0, 1, \cdots, \ni$, $\sum x_i = \nu$ 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:

$$h(x) = 1$$

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

$$T(x) = \log p$$

$$B(\theta) = N \left(\sum_k \log \Gamma(\alpha_k) - \log \Gamma \left(\sum_k \alpha_k \right) \right)$$

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 [], 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 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:

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

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

Ringraziamenti

Bibliografia

[1] Blei David M., Ng Andrew Y., Jordan Michael I. *Latent Dirichlet Allocation.* Journal of Machine Learning Research, 2003.

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