Indice

Introduzione

Questo lavoro di tesi si propone lo scopo di progettare ed implementare un programma in grado di acquisire, processare e analizzare 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 stumenti, 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 dei topic model 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 un file di testo, sviluppando metodi e modelli statistici per l'estrapolazione dei termini-chiave in modo da clusterizzare un certo numero di documenti secondo l'argomento trattato; in secondo luogo, si affronta la questione di gestire *big data*, 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.

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. Par-

ticolare 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'algortimo 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 dei social media: si esegue il codice su una selezione di notizie provenienti da alcuni siti di testate giornalistiche.

I codici usati si trovano in appendice, tuttavia l'intero materiale è gratuitamente disponibile e fruibile in rete, all'indirizzo:

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.

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 modo più efficente possibile. Sembrerebbe che la crescita della tecnologia

[1] 1 Terabyte = 1000 Gigabyte

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. 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.

1.3 La scelta del caso di studio

1.4 Possibili generalizzazioni

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 e l'analisi dei Big Data

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 Il contesto dei Big Data

3.2 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.

3.2.1 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

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

registrata.

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.

3.2.2 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.

3.3 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 [].

3.3.1 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).

3.3.2 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.

3.4 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 intefacciarsi 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.

3.4.1 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.

3.4.2 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.

3.4.3 *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.

3.4.4 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

Impostazione del lavoro

Capitolo 5

Caso di Studio: Risultati

Appendice A

Codice utilizzato.

Appendice B

Alcune distribuzioni di probabilità (?).

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.