Sound of Data

Riccardo Cervero 794126
Marco Ferrario 000000
Pranav Kasela 846965
Federico Moiraghi 799735

Università degli Studi di Milano Bicocca

Anno Accademico 2018/19

Obiettivo del progetto è analizzare la discussione mediatica riguardante i soggetti del mondo musicale contemporaneo e passato, ottenendo, grazie alla gestione in tempo reale di un flusso di tweets, gli artisti, le produzioni e i generi citati all'interno del testo, in modo da produrre uno schema a grafo in grado di rappresentare efficacemente la composizione dei topic musicali nell'arco del tempo, e osservare quali legami possano esistere fra le comunità formate da utenti che scrivono degli stessi argomenti.

Indice

Parte I
Introduzione

I traguardi posti dal progetto *Sound of Data*[1], hanno innanzitutto richiesto la raccolta di un volume di dati sufficienti per costruire un *knowledge graph* adeguato alla materia. Scaricato un *dump* di `musicbrainz.org` come database relazionale - già esportato in formato *Tabular Separated Values* dai manutentori-, si è dapprima importato in Apache Hadoop[2] per effettuare una rapida pulizia preliminare e infine esportato in modo tale da costruire un grafo Neo4J[3]. Costituita quindi la base di conoscenza su cui operare, i *tweet* sono stati raccolti in tempo reale grazie ad Apache Kafka[4]. Durante questa fase di *streaming*, essi subiscono varie operazioni di *preparation* e vengono filtrati da un rudimentale strumento di *instance matching*. Grazie ad un efficiente plug-in che permette all'utente di sfruttare il *DBMS Neo4j* identicamente ad un qualsiasi *Consumer* di *Apache Kafka*, il risultato si costituisce automaticamente in uno schema a grafo. Dopo la complessa fase di gestione e raccolta dei dati, è stato possibile condurre un'analisi finale sulle dinamiche della discussione musicale che si instaurano fra i singoli utenti all'interno di comunità astratte ed estremamente mutevoli, e sulle relazioni, più o meno intense, esistenti fra di esse. Le progressiva composizione che i nodi associati ai tweet assumono all'interno del grafo viene studiata ad intervalli regolari, corrispondenti alle quattro fasce orarie relative al mattino, pomeriggio, sera notte. Si è preferito adoperare tale ciclo giornaliero, anzichè un più ampio ciclo settimanale, o addirittura mensile per due ragioni precise. Innanzitutto, esso si rivela migliore dal punto di vista interpretativo: risulta di estremo interesse osservare i trend ricorrenti nei vari momenti della giornata, piuttosto che rifersi all'arco settimanale. Inoltre, il ciclo giornaliero ha il vantaggio di non risentire della distorsione provocata dagli eventi, siano essi previsti o imprevisti. Infatti, questi sono in grado di assorbire l'interessa della gran parte dell'opinione pubblica per interi giorni e settimane, penalizzando l'analisi e concentrando tutti gli utenti verso un numero limitatissimo di argomenti musicali. Due esempi riguardanti tale problematica, e le sue conseguenze negative, possono essere citati per quanto concerne il genere "*drone*". Durante il periodo di raccolta, si sono succeduti due eventi particolarmente rilevanti: l'abbattimento di un drone che ha scatenato nuove tensioni fra Stati Uniti e Iran[5] e uno spettacolo di droni luminosi a Torino[6]. Questi due eventi, raccogliendo la preoccupazione o l'interesse di centinaia di utenti, hanno fatto sì che il cluster del genere "*drone*", la cui parola presenta inoltre un evidente difetto di polisemia, fosse sproporzionato rispetto alle dimensioni delle altre comunità, se osservato nella settimana e nei giorni degli avvenimenti. Tuttavia, se vista su un arco giornaliero, questa distorsione non risulta eccessivamente grave. Per evitare che altri eventi, seppur riferiti effettivamente al mondo della musica, potessero nuocere alla bontà dello studio, si è pertanto deciso di adoperare queste quattro fasce orarie come intervalli temporali di riferimento.

[1] `https://github.com/pkasela/Sound-of-Data`
[2] `https://hadoop.apache.org`
[3] `https://neo4j.com`
[4] `https://kafka.apache.org`
[5] `http://www.ansa.it/sito/notizie/mondo/mediooriente/2019/06/20/iran-pasdaran-abbattuto-drone-usa_b8922ea0-e61c-436b-8f66-60f3be32ed21.html`
[6] `https://www.guidatorino.com/eventi-torino/san-giovanni-torino-droni-2019/`

Parte II
Costruzione dello *knowledge graph*

1 Data Cleaning con Python e Pig

I dati vengono scaricati dal sito di musicbrainz attraverso il file *get_data.py*[7], e, sempre al suo interno, gestiti preliminarmente, in modo da sostituire, attraverso il comando sed, tutti i caratteri "
N" con campi vuoti in tutti i file, e salvare solamente i file necessari alla creazione del *database* a grafo. La lista dei generi è anch'essa presente sul sito di *musicbrainz*. Pertanto, questa viene estratta e salvata all'interno del codice attraverso uno scraper. Poi, grazie al confronto con il file *tag*, vengono individuati e rimossi tutti i *tag* che non rappresentano un genere musicale.

I file vengono successivamente trasferiti e processati all'interno del *file system* HDFS. Nel dettaglio, questa fase viene eseguita attraverso l'utilizzo di Apache Pig, una piattaforma atta proprio all'elaborazione dei dati. La decisione di utilizzare Pig anzichè SQL dipende principalmente dal fatto che il caricamento dei dati in un database relazionale sia molto più lenta, e dalla capacità di Pig di riuscire a sfruttare a pieno la potenza di HDFS. L'*engine* adoperato in combinazione con Pig è Tez, che rende molto più veloce il *processing* dei dati, saltando diverse fasi di scrittura dei risultati parziali su HDFS. Apache Tez è costruito direttamente sopra YARN, e migliora drasticamente la velocità rispetto a *MapReduce*, mantenendo la sua stessa scalabilità.

Più precisamente, la velocità di esecuzione è ottenuta grazie all'invio diretto dei dati da un processo all'altro, evitando la scrittura all'interno del file system, eccezion fatta per

[7] `https://github.com/pkasela/Sound-of-Data/blob/master/musicbrainz_data/Data_Cleaning/get_data.py`

hadoop2.png

Figura 1: *Differenza architettura di HADOOP 1.0 con MR e 2.0 con Tez*

tez.png

Figura 2: *Architettura di HADOOP 2.0 con Tez*

i checkpoints. Inoltre, la definizione del *job* sfrutta i *Directed Acyclic Graph*, dove i vertici rappresentano gli step del processo e gli archi la connessione tra i vertici *Producer* e *Consumer*.

Conclusa la fase di pulizia, i dati vengono trasferiti all'esterno di HDFS, sul filesystem, e i risultati di Pig sono uniti mediante il comando *cat* di *Linux*. A questo punto i dati sono pronti per essere importati in Neo4j.

2 Import dei dati in Neo4J

Infine, il risultato dell'elaborazione svolta da Pig viene caricato all'interno del *DBMS Neo4j*, usando neo4j-import, aggiungendo degli indici sui *gid* delle varie entità, in modo da velocizzare la ricerca da svolgere successivamente.

Parte III
Analisi dei tweet

3 Elaborazione mediante Apache Kafka

Nella fase di analisi in tempo reale, i tweet vengono filtrati ed elaborati mediante una sequenza di operazioni, attraverso una *pipeline* costituita dalla concatenazione di due coppie di processi Producer e Consumer. L'architettura di elaborazione in *streaming* è stata configurata in questa maniera affinchè garantisse una maggiore scalabilità ed una migliore performance, riuscendo a reggere più alti volumi di *tweet* in ingresso. Questi vengono estratti grazie all'API *Tweepy*[8] per Python - messa a disposizione da Twitter -, utilizzando come parole chiave di ricerca 400 generi musicali fra i 419 ottenuti dallo scraping della pagina *'https://musicbrainz.org/genres'*, e filtrando soltanto i testi pubblicati in lingua italiana, e inviati come messaggi di un *Producer* di Kafka, inizializzato mediante la libreria *Kafka-python*[9], ad una prima Topic. All'interno di un secondo file[10], associato ad un nuovo processo Producer, dunque ad una seconda Topic connessa in *streaming* con la precedente, il JSON grezzo che costituisce ciascun "tweet" in ingresso viene processato in vari step dalla funzione *tweet_preparations*. La prima fase coincide con l'estrazione dello "*screen name*" dell'utente, del contenuto postato e dell'ora e data precisa di pubblicazione. In particolare, il testo viene modificato in modo da rendere leggibili i caratteri speciali e i cosiddetti *escape characters*. La seconda fase sfrutta l'API di *Botometer*[11] per calcolare la probabilità - definita "*score*" - che un profilo sia in realtà gestito da un BOT, osservandone il comportamento passato. Poiché questo tipo di gestione automatizzata degli account altera, spesso fortemente, la discussione mediatica, aumentando arbitrariamente il flusso di determinati contenuti, si è deciso di non archiviare i tweet appartenenti a tali profili. Nel dettaglio, se lo score associato all'utente di ogni tweet supera una soglia fissata al 90%, lo *screen name* viene memorizzato all'interno di una *blacklist*, altrimenti smistato in una *whitelist*, in modo che il programma possa riconoscere più velocemente ogni profilo ed evitare inutili ricalcoli. Per aumentare l'efficienza di questo storage e garantire la scalabilità, si è scelto di adoperare il *data store Riak*, di tipo *NoSQL Key-Value*, attraverso la propria libreria in Python[12]. La terza fase verifica che il testo sia effettivamente legato al tema musicale. La quarta e ultima fase consiste nel riconoscimento delle parole riguardanti il contesto musicale, aggiungendo al dizionario prodotto dalle fasi precedenti tre chiavi corrispondenti rispettivamente alla lista di artisti, album o canzoni estratti dal testo analizzato. Questi due step sono permessi dall'uso dell'API fornita dal sito *musicbrainz.org*. Per maggiori dettagli sul processo di analisi semantica, si rimanda alla sezione dedicata. Infine, ogniqualvolta si verificano le due condizioni note, cioè che l'utente non abbia probabilità elevata di essere un BOT e che il testo pubblicato si riferisca effettivamente al mondo della musica, ciascun risultato della funzione *tweet_preparations* viene ricodificato e passato come messaggio al secondo *Producer* di *Apache Kafka* - menzionato in precedenza-, che effettuerà l'invio del dato finale alla *topic* nominata "*KafkaTopic*". Grazie all'utilizzo di questa piattaforma, il flusso di *feed* viene gestito in tempo reale, assicurando bassissima latenza e grande scalabilità.

[8] https://tweepy.readthedocs.io/en/latest/

[9] https://pypi.org/project/kafka-python/

[10] https://github.com/pkasela/Sound-of-Data/blob/master/NL/Kafka_Producer_2.py

[11] https://rapidapi.com/OSoMe/api/botometer/details

[12] https://github.com/basho/riak-python-client

4 Consuming dei Tweets in Neo4j

Un principale vantaggio di *Apache Kafka* consiste nella capacità di connettersi efficientemente a sistemi esterni, garantendo in tal modo la continuità dello *stream* di dati da una fonte - la class *Listener* adoperata in Python per "ascoltare" i tweet grazie all'API Tweepy -, ad una "*landing zone*", che in questo caso corrisponde al DBMS *Neo4j*. Si è scelto di effettuare lo *storage* dei tweets in un *GraphDB*, e in particolare *Neo4j*, tra le varie motivazioni, per le sue caratteristiche di estrema scalabilità, efficienza nella gestione, elevata capacità di adattamento al fenomeno di studio e interpretabilità dei dati. L'esportazione diretta dei tweets da *Apache Kafka* all'interno del *data store* avviene grazie al *plug-in* di *Neo4j* chiamato "*Neo4j Streams Consumer*". Questo si configura come un'applicazione che effettua un'ingestione diretta e automatizzata all'interno di *Neo4j*, permettendo la lettura dei dati presenti nella *topic* identicamente a qualsiasi applicazione *Consumer* di *Kafka*. *Neo4j Streams Consumer* permette all'utente di specificare arbitrariamente le relazioni, entità e proprietà in cui i *payloads* di *Apache Kafka*, corrispondenti al JSON di ciascun tweet importato, dovranno essere organizzati per costruire progressivamente il grafo. La struttura di quest'ultimo viene dichiarata attraverso un *Cypher template*, ovvero un insieme di *queries* semantiche di *Cypher*, all'interno di un file di configurazione, che nel caso presentato è il "*docker-compose.yml*"[13], poiché il progetto viene eseguito mediante *Docker Desktop*, in quanto quest'ultimo presenta il vantaggio di riuscire a far comunicare automaticamente e in maniera trasparente i containers coinvolti. Una volta specificato il *Cypher template*, installato il *plugin* all'interno del *Docker*, e successivamente montati i container e i relativi collegamenti, la fase di mera esecuzione del progetto avviene mediante la funzione denominata "*streams.consume*", la quale crea immediatamente la struttura del grafo come indicato. Nello specifico, all'arrivo di un tweet verranno dunque aggiunti i nodi "*Tweet*","*User*" ed un terzo che riporta ora e data di pubblicazione. Il primo conterrà le *property keys* del testo "*text*", e dello "*screen_name*". Il secondo, invece, conterrà solamente la proprietà "*name*", che riporta lo *screen name*. Tra primi i due nuovi nodi verrà generata la relazione "*BELONGS_TO*", ad indicare il profilo di appartenenza di ciascun post, mentre "User" sarà connesso all'ora/data di pubblicazione da un *edge* etichettato come "Twitted". In più, il programma effettuerà automaticamente una *merge* della coppia appena generata con i nodi già presenti all'interno del grafo, importati precedentemente da *musicbrainz.org*, nella seguente maniera: quando il testo cita, ad esempio, un artista, il software crea la relazione "*TALKS_ABOUT*", rappresentata da un arco connesso al nodo a cui appartiene il "*value*" di quel preciso artista - cioè il suo id -, il quale sarà già a sua volta collegato ai generi musicali cui appartengono le proprie produzioni. In questo modo, il nodo riferito al genere costituirà il centro di ogni cluster di utenti. Lo stesso accade per quanto riguarda le *labels* relative al titolo di una canzone, oppure di un album. A questo punto, il grafo apparirà direttamente cosparso di nodi utente, raggruppati in comunità a seconda dell'oggetto dei propri tweet, ingrandendosi e variando in base alle dinamiche interne alla discussione mediatica nell'arco di vari cicli orari e giornalieri. Grazie a questa semplice struttura, l'analisi avviene già in maniera grafica e trasparente, e l'utente è in grado di distinguere la grandezza di queste comunità variabili, intendere quali legami esistano fra di esse, e quanto questi siano intensi. Per ottenere queste informazioni, è infatti sufficiente scaricare un *dump* del grafo a intervalli regolari e confrontare i vari risultati. Il risultato finale, dall'interfaccia web di Neo4j, apparirà come nella Figura 3.

Invece, la composizione dei cluster, delle comunità ben separate, verrà mostrata come nella Figura 4.

[13] `https://github.com/pkasela/Sound-of-Data/blob/master/Neo4j\%20\%26\%20kafka/docker-compose\%20configuration.yml`

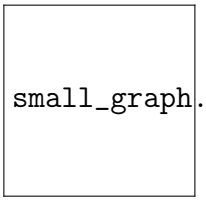

Figura 3: *Configurazione della struttura del grafo*

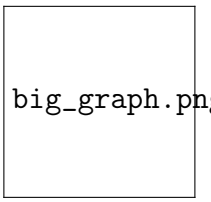

Figura 4: *Composizione parziale delle comunità all'interno del grafo*

5 Riconoscimento delle istanze nel testo

Data la mole di dati scaricabili, si è deciso di costruire un strumento di *instance matching* creato *ad hoc* per i tweet. Vista la scarsa abilità degli algoritmi basati su reti neurali e *deep learning* nell'analisi del breve (e quindi decontestualizzato) testo di un tweet, si è costruito un modello per l'identificazione di ipotetiche entità su cui basarsi per confronti col database, sfruttando i *gid*, i codici identificativi di musicbrainz disponibile grazie all'API[14] offerta da `musicbrainz.org` stesso).

5.1 Identificazione delle entità

Le entità sono riconosciute non mediante *machine learning* ma grazie a semplici stratagemmi linguistici. Prima di tutto il testo del tweet è ripulito da eventuali abbreviazioni gergali; subito dopo sono ricercate le parole nel testo che non risultano essere italiane: analizzando tweet in lingua italiana, si presume che una stringa in lingua diversa abbia una certa importanza. Per fare questo è usato un mero correttore ortografico che evidenzia quali parole non sono riconosciute; di aiuto nel compito è anche una semplice espressione regolare che tenta di stabilire quali parole non seguano la costruzione sillabica italiana (rientrando quindi o nella categoria dei sostantivi della quinta classe o, nei casi fortunati, nelle entità cercate): secondo le regole linguistiche, una sillaba correttamente formata è composta da un numero massimo di tre consonanti seguita da una vocale e da al più una sola consonante - o vocale con suono consonantico, formando quindi un dittongo. Alle entità così individuate si aggiungono tutte le parole scritte in maiuscolo - che in un testo di così bassa formalità non sempre coincidono coi nomi propri-, anche se a inizio frase. Grazie all'uso della punteggiatura - le virgolette - e delle preposizioni, si tenta inoltre di stabilire se l'entità rilevata è un presunto autore o una presunta opera. Per individuare gli id delle entità riconosciute con il metodo descritto precedentemente, è stato utilizzato Python con la libreria *Musicbrainzngs*. Attraverso le funzioni *search* è possibile consultare il database sfruttando un server di ricerca costruito utilizzando la tecnologia Lucene. *Musicbrainz* è interrogato con la funzione search_artists per le entità riconosciute come autori, search_recordings per quelle individuate come canzoni, ed entrambe, con in aggiunta search_release_groups (funzione relativa alla ricerca degli album), per quelle la cui tipologia è incerta. La problematica principale emersa in questa parte è stata la difficoltà ad attribuire all'entità analizzata l'id che la caratterizza effettivamente. Sono infatti molto frequenti i casi di omonimia tra canzoni ed album relativi ad artisti differenti ed è quindi possibile che il risultato ottenuto non sia quello desiderato. Per cercare di aumentare le probabilità di restituire i gid corretti è stato impostato un valore di similarità pari a 0.95 nell'interrogazione al database, inoltre per le canzoni e per gli album si è andato a verificare se ci fossero corrispondenze con le entità artisti precedentemente individuate nel testo.

[14] `https://python-musicbrainzngs.readthedocs.io/en/v0.6/`

5.1.1 Prestazioni del modello e margini di miglioramento

1. Casi di polisemia: drone (già citato), dance, club, opera. 2. Problemi con la lingua 3. trap in cui compare anche ['gay','sissy','femboy','daddy','femboi']

Parte IV
Visualizzazioni dei dati

Parte V
Risultati e conclusioni