Sound of Data

Riccardo Cervero 794126
Marco Ferrario 000000
Pranav Kasela 000000
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.

Indice

Parte I
Introduzione

Per raggiungere i traguardi posti dal progetto *Sound of Data*[1], si son dovuti raccogliere 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]. Costruita quindi la base di conoscenza su cui operare, si sono raccolti *tweet* in tempo reale grazie ad Apache Kafka[4], per poi analizzarli in automatico con un rudimentale strumento di *instance matching*.

[1] `https://github.com/pkasela/Sound-of-Data`
[2] `https://hadoop.apache.org`
[3] `https://neo4j.com`
[4] `https://kafka.apache.org`

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, che oltre a scaricare i dati effettua un'analisi preliminare di sostituire tutti i caratteri "
N" con campi vuoti in tutti i file attraverso il comando sed e salva solo i file necessari per il database a grafo che si vuole creare. La lista dei generi è presente sempre sul sito di musicbrainz, con uno scraper viene salvato nel codice e confrontato con il file tag e vengono eliminati tutti i tag che non sono dei generi.

I file vengono trasferiti su HDFS per essere processati, in particolare usiamo Apache Pig, che è una piattaforma per processare i dati, la decisione di utilizzare Pig piuttosto che SQL è dovuta al fatto che il caricamento dei dati in SQL è molto lenta e inoltre Pig riesce a sfruttare la potenza di HDFS. L'engine usato con Pig è Tez, che rende molto più veloce l'esecuzione del processamento dei dati saltando diverse fasi di scrittura dei risultati parziali su HDFS.

Una volta finita la pulizia i dati vengono trasferiti fuori da HDFS sul filesystem, e vengono unite i risultati di pig con il comando cat. A questo punto i dati sono pronti per essere caricati su Neo4j.

2 Neo4J

I dati puliti da Pig vengono caricati su neo4j usando neo4j-import.

Parte III
Analisi dei tweet

3 Elaborazione mediante Apache Kafka

Nella fase di analisi in tempo reale, i tweet vengono filtrati ed elaborati attraverso una sequenza di operazioni. Innanzitutto, essi vengono estratti grazie all'API *Tweepy* 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. Poi, all'interno della class *Listener*, la funzione *tweet_preparations* elabora in vari step il JSON di ogni tweet ricevuto. 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* 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. 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, nel caso in cui si verifichino 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, ogni risultato della funzione *tweet_preparations* viene ricodificato e passato come messaggio ad un *producer* di *Apache Kafka*, inizializzato mediante la libreria *Kafka-python*, ed inviato alla *topic* nominata "*Music_Tweets*". Grazie all'utilizzo di questa piattaforma, il flusso di *feed* viene gestito in tempo reale, assicurando bassissima latenza e grande scalabilità.

4 Consuming dei Tweets in Neo4j

Un principale vantaggio di *Apache Kafka* è la capacità di connettersi efficientemente a sistemi esterni, garantendo in tal modo la continuità dello stream di dati da una fonte - il *Listener* di Python -, 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*", 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*", 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*" e "*User*". Il primo conterrà le seguenti *property keys*: il testo "*text*", lo "*screen_name*" e "*created_at*, cioè ora e data di pubblicazione. Il secondo, invece, conterrà solamente la proprietà "*name*", che riporta lo *screen name*. Tra i due nuovi nodi verrà generata la relazione "*BELONGS_TO*", ad indicare il profilo di appartenenza di ciascun post. 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. 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 segue: Screenshot

5 Riconoscimento delle istanze nel testo

Data la mole di tweet scaricabili, si è deciso di costruire un strumento di *instance matching* creato *ad hoc* per i tweet. Vista la scarsa abilità di algoritmi basati su reti neurali e *deep learning* ad analizzare il 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 (grazie alla API[5] 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 seguono 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 e quindi cercata all'interno del database.

5.1.1 Prestazioni del modello

5.1.2 Margini di miglioramento

5.2 Riconoscimento delle entità

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

Parte IV
Visualizzazioni dei dati

Parte V
Risultati e conclusioni