Sound of Data

Riccardo Cervero **794126**
Marco Ferrario **795203**
Pranav Kasela **846965**
Federico Moiraghi **799735**

Università degli Studi di Milano Bicocca

Anno Accademico 2018/19

Obiettivo del progetto è analizzare la discussione mediatica sul social network Twitter, studiandone in tempo reale l'andamento nell'ambito musicale, presente e passato. Lo scopo è verificare la presenza di variazioni all'interno del ciclo giornaliero e quella delle comunità digitali, e le relative interconnessioni. Il presente lavoro può essere di aiuto nella costruzione di sistemi di raccomandazione interni a Twitter (quali pagine consigliate) e nella programmazione di BOT, a scopo pubblicitario, che intendano entrare in una discussione mediatica già presente (ovvero argomenti di tendenza) per inserire i propri messaggi.

Indice

Parte I
Introduzione

I traguardi posti dal progetto *Sound of Data*[1], hanno innanzitutto richiesto l'uso di uno *knowledge graph* adeguato alla materia: scaricato un *dump* di `musicbrainz.org`, versione *semantic web* di `last.fm`, disponibile pubblicamente in formato `.tsv` (*Tabular Separated Values*), si è provveduto ad importarlo all'interno di Apache Hadoop[2] per effettuare una rapida pulizia preliminare. Fatto ciò, i dati sono stati importati in Neo4J[3], grazie al pratico programma `neo4j-import`, in modo da costruire lo *knowledge graph* richiesto per l'analisi dei *tweet*, raccolti ed analizzati in tempo reale grazie ad Apache Kafka[4]: grazie ad un efficiente plug-in[5] che permette l'uso di *Neo4j* come *Consumer*, il singolo tweet è archiviato nativamente in uno schema a grafo. Durante il processo di *streaming*, questi subiscono varie operazioni di *preparation* e vengono filtrati da un rudimentale strumento di *instance matching*.

Dopo la complessa fase di gestione e raccolta dei dati, è stato possibile condurre un'analisi finale sulle dinamiche della discussione musicale instauratesi 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 dei nodi, rappresentanti i *tweet* all'interno del grafo, è studiata ad intervalli regolari, corrispondenti a quattro fasce orarie relative al mattino, pomeriggio, sera e notte. Si è preferito adoperare tale ciclo giornaliero, anziché un più ampio ciclo settimanale o addirittura mensile, per due ragioni precise. Innanzitutto, questo garantisce una più facile interpretazione: risulta di estremo interesse osservare i trend ricorrenti nei vari momenti della giornata, piuttosto che riferiti all'arco settimanale. Inoltre, il ciclo giornaliero ha il vantaggio di non risentire della distorsione provocata dagli eventi, previsti o meno. Infatti, essendo gli eventi in grado di assorbire l'interesse della gran parte dell'opinione pubblica per interi giorni e settimane, questi penalizzano l'analisi e concentrano tutti gli utenti verso un numero limitato di argomenti musicali. Due esempi riguardanti tale problematica, e le sue conseguenze negative, possono essere citati per quanto concerne il genere *drone*, per cui, durante il periodo di raccolta, si sono succeduti due eventi particolarmente rilevanti: il primo riguarda l'abbattimento di un drone americano in territorio Iraniano, causando nuove tensioni fra i due Paesi[6], mentre il secondo consiste in uno spettacolo di droni luminosi a Torino[7] per la festa patronale.

Raccogliendo l'attenzione di centinaia di utenti, i due eventi e la polisemia della parola hanno fatto sì che la comunità di ascoltanti il genere musicale *drone* fosse sproporzionatamente più grande rispetto alle altre nei giorni degli eventi; ma tale distorsione è stata diluita dalla divisione in fasce orarie e dalla moltitudine dei giorni in cui è stata effettuata la raccolta dati.

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

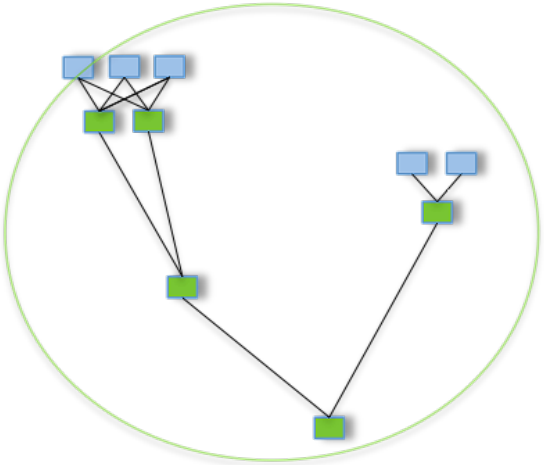
Parte II
Costruzione dello *knowledge graph*

1 Data Cleaning con Python e Pig

I dati sono stati scaricati dal sito di musicbrainz attraverso lo script python `get_data.py`[8] che cerca sul sito l'ultima versione del database e la scarica col comando `wget` dei sistemi Unix. Decompressa quindi la cartella con il dump del database e selezionati solamente i file necessari allo scopo, vengono eseguite delle operazioni `cat-sed` per effettuare una rapida pulizia preliminare (la gestione dei valori nulli `\N`). È stata salvata inoltre una lista di generi musicali, già presente in una pagina HTML del sito, tramite uno *scraper*, che ne analizza l'elenco per importarla in Python.

I file sono stati successivamente importati e processati all'interno del *file system* HDFS. Nel dettaglio, questa fase viene eseguita attraverso l'utilizzo di Apache Pig, piattaforma atta all'elaborazione di grosse quantità di dati. La decisione di utilizzare Pig anzichè SQL dipende principalmente dalle migliori prestazioni del primo nella fase di caricamento, e dalla maggiore versatilità della suite Hadoop per la fase di elaborazione. L'*engine* adoperato in combinazione con Pig non è MapReduce ma Tez: costruito sopra YARN[1], migliora drasticamente le prestazioni mantenendo la stessa scalabilità poiché evita alcune fasi di scrittura di risultati parziali sull'HDFS (vedere Figura 1 per maggiori dettagli[4]).

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 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* (come mostrato in figura[1, 4]).



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

2 Import dei dati in Neo4J

In seguito, il risultato dell'elaborazione svolta da Pig viene caricato all'interno del *DBMS Neo4j* usando il comando `neo4j-import`[2] e indicizzando i *gid* delle varie entità, in modo da velocizzare la ricerca da svolgere successivamente.

Il risultato è un grafo dal peso approssimativo di 5Gb contenente i vari artisti, i relativi album e le relative case discografiche, collegati tra di loro in modo opportuno. Si è preferito ridurre la capillarità del database evitando di inserire le singole canzoni contenute negli album, sia perché una simile precisione sarebbe stata difficile da gestire in fase di analisi, sia a causa della mancanza nel dump del database di tale informazione (comunque presente nella versione online), che avrebbe richiesto un tempo superiore alla settimana (dovuto perlopiù alla latenza supportata dal sito) per poter essere recuperata.

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

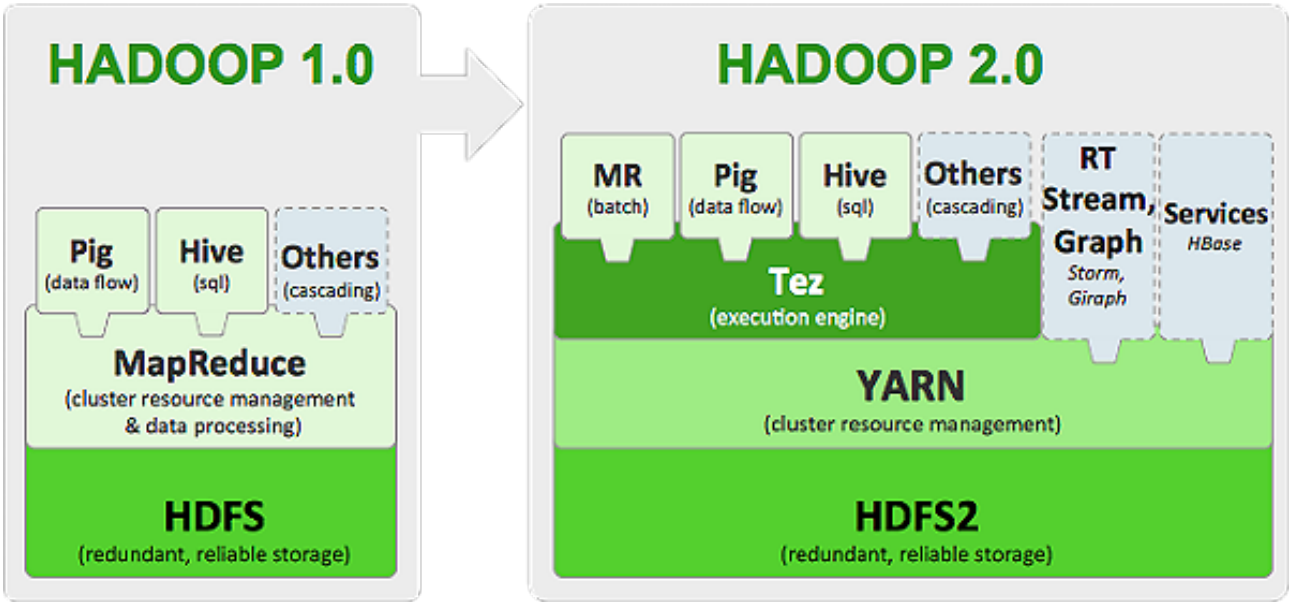


Figura 1: *Differenza architteturale di HADOOP 1.0 con MapReduce rispetto alla versione 2.0 con Tez.*

Parte III
Streaming dei tweet

3 Elaborazione mediante Apache Kafka

Nella fase di gestione e 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 così configurata per garantire una maggiore scalabilità ed una migliore performance, riuscendo a reggere volumi di *tweet* in ingresso maggiori. Questi vengono estratti grazie all'API *Tweepy*[9] per Python, utilizzando come parole chiave di ricerca una selezione di 400 generi musicali fra i 419 ottenuti dallo scraping della pagina `'https://musicbrainz.org/genres'`, e ricevendo soltanto i testi pubblicati in lingua italiana.

Quindi i *tweet* vengono inviati ad una prima Topic come messaggi di un *Producer* di Kafka, inizializzato mediante la libreria *Kafka-python*[10]. All'interno di un secondo file[11], 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*[12] per calcolare la probabilità - definita *score* - che un profilo sia in realtà gestito da un BOT, osservandone il comportamento passato. Poiché i profili gestiti in modo automatico alterano, spesso fortemente, la discussione mediatica aumentando arbitrariamente il flusso di determinati contenuti, si è deciso di non

[9] `https://tweepy.readthedocs.io/en/latest/`

[10] `https://kafka-python.readthedocs.io/en/master/`

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

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

archiviare i *tweet* appartenenti a tali profili. Nel dettaglio, se lo score associato all'utente di ogni *tweet* supera una soglia (fissata arbitrariamente al 90%), lo *screen name* viene memorizzato all'interno di una *blacklist*, altrimenti viene smistato in una *whitelist*, in modo che il programma possa riconoscere più velocemente ogni profilo ed evitare inutili ricalcoli. In questo modo, si riduce il numero di richieste fatte all'API, gratuita fino ad un certo volume e poi a pagamento. Per aumentare ulteriormente l'efficienza di questo processo e garantire la scalabilità, si è scelto di adoperare il *data store Riak*, di tipo *NoSQL Key-Value*, interfacciato per mezzo della propria libreria Python[13][5]: viene così archiviato lo *score* di ogni utente e la richiesta all'API è effettuata solamente se non si trova la chiave nel database.

La terza fase verifica che il testo contenga entità di rilevo, grazie ad un rudimentale *istance matcher* che tenta di riconoscere le 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. Ciò è permesso dall'uso dell'API[14] fornita dal sito *musicbrainz.org* e basata su Apache Lucene[15]. Per maggiori dettagli si rimanda alla sezione dedicata.

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` è 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* è gestito in tempo reale, assicurando bassissima latenza e grande scalabilità.

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

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

[15] `https://lucene.apache.org/`

4 Consuming dei Tweets in Neo4j

Uno dei principali vantaggi 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 classe *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* per le sue caratteristiche di estrema scalabilità, efficienza nella gestione dei dati, elevata capacità di adattamento al fenomeno di studio e facile interpretabilità. L'esportazione diretta dei tweet da *Apache Kafka* all'interno del *data store* avviene grazie a un *plug-in* di *Neo4j*, "*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* allo stesso modo di qualsiasi applicazione *Consumer* di *Kafka*.

Neo4j Streams Consumer permette all'utente di specificare arbitrariamente 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 `docker-compose.yml`[16], 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 *plug-in* 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

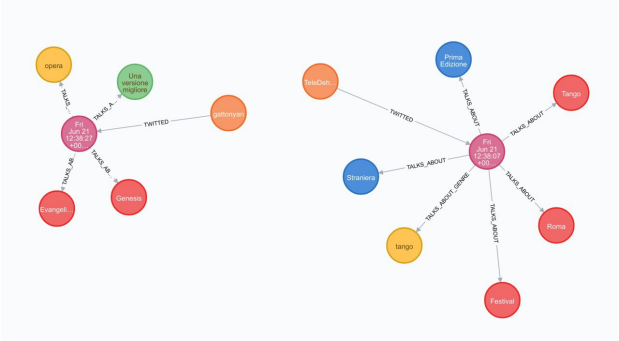
[16] `https://github.com/pkasela/Sound-of-Data/blob/master/Neo4j\%20\%26\%20kafka/docker-compose\%20configuration.yml`

immediatamente la struttura del grafo come indicato.

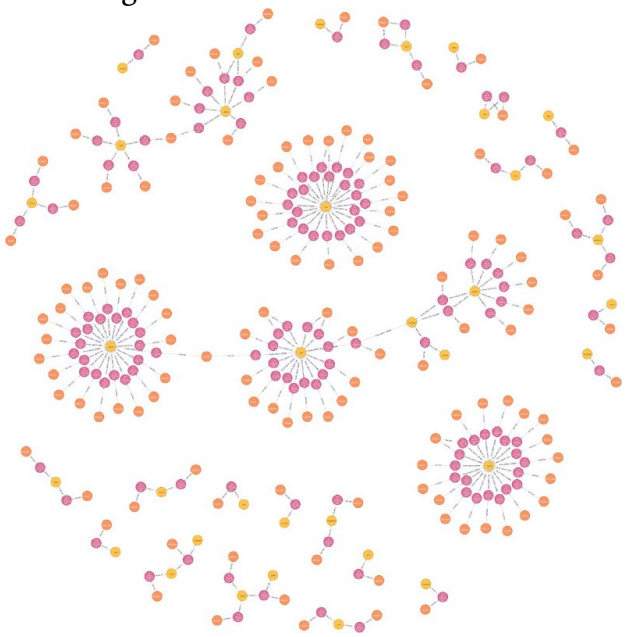
Nello specifico, all'arrivo di un *tweet* verranno dunque aggiunti i nodi `Tweet`, `User` ed un terzo che riporterà ora e data di pubblicazione. Il primo conterrà le *property keys* del testo `text`, e dello `screen_name`. Il secondo, invece, conterrà solamente la proprietà "*name*", che riporta lo *screen name*. Tra i primi due nuovi nodi verrà generata la relazione `BELONGS_TO`, ad indicare il profilo di appartenenza di ciascun post, mentre `User` sarà connesso alla data e ora di pubblicazione da un *edge* etichettato come `Twitted`.

Inoltre il programma effettua automaticamente un'operazione di *merge* della coppia appena generata coi nodi già presenti nel grafo: quando è individuato l'`id` di un'entità presente nel testo, il software collega il nodo del tweet all'entità con una relazione `TALKS_ABOUT`. L'entità presente su musicbrainz viene collegata al proprio genere musicale, così da fare da tramite tra i nodi riguardanti genere e *tweet*, punto focale del progetto.

Dunque il grafo conterrà una serie di nodi utente, raggruppati in comunità attorno ad artisti o generi a seconda dell'oggetto dei propri *tweet*. Grazie a questa semplice struttura, l'analisi può avvenire già in maniera grafica e trasparente, e l'utente è in grado di distinguere la grandezza di queste comunità, ed intendere quali e quanto intensi legami esistano fra di esse. 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 seguente figura:



Invece, la composizione dei cluster, cioè delle comunità ben separate, verrà mostrata come di seguito:



5 Riconoscimento delle istanze nel testo

Data la quantità e qualità dei dati, si è deciso di costruire un strumento di *instance matching* creato *ad hoc* per i tweet: le API disponibili basate su *deep learning* sono spesso allenate su altri tipi di testo (articoli di giornale o opere letterarie) e spesso non offrono una velocità adeguata per l'analisi in tempo reale. Il problema della qualità del testo è fondamentale: serve uno strumento che riesca ad analizzare il testo e a trovare le entità corrispondenti senza bisogno di contestualizzazione. Per tale scopo, filtrati già in partenza i tweet grazie a parole chiave a tema musicale, si costruisce un modello che identifichi le istanze restituendone gli `id` grazie all'API di musicbrainz disponibile per Python.

5.1 Identificazione delle entità

Le entità sono riconosciute non mediante *machine learning* ma grazie a semplici stratagemmi linguistici e l'aiuto di un correttore ortografico. Prima di tutto il testo del tweet è ripulito da eventuali abbreviazioni gergali (che vengono sostituite dalla propria forma estesa),

link e caratteri speciali. Subito dopo sono ricercate le parole nel testo che non risultano essere italiane: analizzando tweet in lingua italiana, il cui tema è sicuramente musicale, si presume che una stringa in lingua diversa abbia una certa importanza; ciò è fatto sia analizzandone l'ortografia sia verificando con un'espressione regolare che le parole rispecchino la regola di costruzione delle sillabe in italiano[17]. In tal caso, la parola è un sostantivo di una lingua straniera e come tale quindi degna di attenzione. Infine sono analizzate le parole la cui iniziale è maiuscola, col supporto di un albero sintattico (costruito grazie ad un programma esterno, `TreeTagger`[18], interfacciato con Python grazie ad un wrapper[19] fornito dagli stessi sviluppatori) per verificare che siano nomi propri.

Inoltre, grazie all'uso della punteggiatura e delle preposizioni, si tenta di stabilire se la presunta entità sia un artista o un'opera, verificando poi con certezza confrontando con lo *knowledge graph*.

Tramite le API di musicbrainz, attraverso una serie di funzioni `search`, è possibile consultare il database online utilizzando la tecnologia *Lucene*. Analizzate le entità e tentato di stabilire cosa rappresentino, si interroga la base di conoscenza con le funzioni `search_artists` e `search_recordings` per le entità riconosciute rispettivamente come artisti e opere, a cui si aggiunge `search_release_groups` (funzione relativa alla ricerca degli album), per quelle di tipologia incerta. Per verificare la corrispondenza tra entità e risultati di ricerca, si è scelta la politica di accettare il risultato la cui similarità coseno è maggiore, a patto che superi una soglia arbitraria di 0.95. Inoltre, per aumentare la precisione, si sono incrociati i risultati parziali delle ricerche per testare se nello stesso testo compaia l'opera accompagnata dal proprio autore.

[17] Una sillaba in italiano è composta da un massimo di tre consonanti iniziali, vocale ed eventuale consonante finale.

[18] `https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/`

[19] `https://perso.limsi.fr/pointal/doku.php?id=dev:treetaggerwrapper`

5.2 Prestazioni del modello e margini di miglioramento

La problematica principale emersa nella costruzione del modello è stata la corretta attribuzione dell'entità nei casi di omonimia: questi casi infatti sono molto frequenti tra canzoni ed album relativi ad artisti differenti ed è quindi possibile che il risultato ottenuto non sia quello desiderato.

Inoltre, nonostante il modello offra una buona *performance* nella selezione dei testi riferiti al mondo musicale e nel riconoscimento delle istanze nel testo, continua a presentare alcune imperfezioni. Un primo difetto deriva dalla difficoltà nella gestione dei casi polisemici: alcuni nomi di genere (in particolare *drone*, *dance*, *club* e *opera*) sono polisemici, distorcendo i valori dell'analisi. Il programma utilizzato è troppo rudimentale per distinguere il contesto e quindi mostra numerosi falsi positivi: il termine *club* in particolare viene spesso citato all'interno di testi riguardanti qualsiasi tipo di associazione; la parola *drone*, come già accennato nell'introduzione, indica, nella maggior parte dei casi, l'aeromobile; l'espressione *dance* è frequentemente slegata alla musica e *opera* possiede troppi significati estranei al contesto in questione.

Un'altra problematica, non legata al modello ma a `tweepy`, è sorta dopo aver rilevato alcuni *tweet* in lingua straniera (perlopiù spagnola) che citavano il termine *trap* senza riferirsi al genere musicale.

Pertanto, il modello raggiunge buoni risultati nell'ambito di questo progetto, rendendo comunque necessari ulteriori tentativi di miglioramento della sua capacità di riconoscere il contesto all'interno del quale le parole vengono citate.

6 Scalabilità

È facile ottenere una buona scalabilità orizzontale del programma grazie alla dimensione fissa (relativamente contenuta) della base di conoscenza e all'architettura dei singoli componenti: Kafka e Riak infatti permettono nativamente la gestione di più *cluster* nel database, distribuendo i dati in modo efficiente. L'unico inconveniente in cui è possibile imbattersi è l'eccessivo aumento dei *tweet* nel tempo: per risolvere questo problema è possibile replicare la base di conoscenza su ogni nodo e in seguito smistare i messaggi tra questi (secondo un criterio prefissato). La struttura a grafo infatti non è frammentabile, ma necessaria principalmente per facilitare la consultazione della base di conoscenza (lo *knowledge graph*) e ottimizzare la gestione dei collegamenti con le entità nei testi. Dunque frammentare le informazioni riguardanti gli utenti è un ottimo sistema per dividere il carico di lavoro fra i *cluster*.

In alternativa, è disponibile una versione a pagamento di Neo4J[3] che permette una divisione multi-cluster del database: in questo modo ogni componente del progetto è scalabile orizzontalmente in modo nativo.

Per comodità, tuttavia, il lavoro svolto è stato fatto interamente sulla stessa macchina, grazie anche al basso volume dei *tweet* ricevuti (dell'ordine di grandezza del Megabyte): ciò ha permesso di evitare tale problematica.

Parte IV
Visualizzazioni dei dati

7 Distribuzione dei principali generi musicali nelle quattro fasce orarie

La prima visualizzazione consiste nella tipologia *barchart*, realizzata con il software *Tableau*. In ascissa, il grafico raccoglie i principali quarantasette generi musicali estratti dall'analisi delle pubblicazioni avvenute tra il 21 e il 28 giugno, ordinati in senso crescente secondo la frequenza assoluta totale. L'ordinata, invece, è segmentata nelle quattro sezioni dedicate rispettivamente al periodo notturno, mattutino, pomeridiano e serale. In ogni sezione, la dimensione della barra corrisponde al numero di volte in cui il dato genere è stato menzionato durante la specifica fascia oraria. Pertanto, sommando verticalmente i valori, si otterrà la dimensione globale associata ad ogni genere.

Tuttavia, sulle misure parziali, relative quindi al singolo intervallo temporale, è stato necessario effettuare un filtro: dato che si è deciso di mantenere la scala naturale dei valori, le barre associate alle frequenze minori di 5 in valore assoluto sono state omesse, per evitare che l'ascissa risultasse carica di un eccessivo numero di generi, peggiorando la consultazione. In ogni caso, avrebbero assunto dimensioni troppo piccole per poter essere visibili.

Perciò, considerando la sola variabile della frequenza assoluta, dal punto di vista dell'"*overview*" il fruitore è in grado di individuare la distribuzione dei vari generi, e intuire immediatamente quali siano i più presenti. Ad esempio, è chiaro come "*dance*" sia il genere comparso più volte durante il pomeriggio, o ancora "*rave*" il più presente durante la mattina. Inoltre, è possibile notare come la notte rappresenti ovviamente il periodo meno

produttivo per quanto riguarda il numero di pubblicazioni, e, viceversa, come la mattina accolga un flusso di tweet a tema musicale nettamente maggiore. Infine, la componente dell'interattività permette di ottenere i dati di ogni singolo genere, approfondendo le sue caratteristiche e valutando la sua disposizione nell'arco della giornata.

Una seconda variabile è stata aggiunta al grafico: la misura di appartenenza di ogni genere musicale ad ogni fascia oraria, ovvero la tendenza percentuale di ogni genere a comparire in una particolare fascia oraria, rispetto al totale dei tweet che lo riguardano. Quanto maggiore è la concentrazione delle citazioni ad esso riferite in uno dei quattro periodi, tanto superiore sarà il suo grado di appartenenza al dato intervallo temporale. L'indice, che, essendo una percentuale, accoglie valori da zero a uno, è rappresentato attraverso il dettaglio della saturazione del colore blu in ogni barra. A questo punto, il fruitore non è soltanto in grado di percepire visivamente la presenza di un particolare genere, ma anche il suo diverso legame con i quattro momenti della giornata. Con l'obiettivo di evidenziare i valori più interessanti sotto questo punto di vista, le prime tre maggiori percentuali di appartenenza, per ogni intervallo temporale, vengono specificate sopra le relative barre. In più, il massimo e il minimo vengono descritti testualmente. È così possibile osservare come la musica classica rappresenti il genere musicale più specifico, poichè citata nel 77% dei casi durante la sera.

Una sezione speciale del grafico, sul lato destro, viene dedicata alle quattro tipologie musicali più presenti. Innanzitutto, questo *zoom* permette al fruitore di cogliere in maniera migliore la differenza e la composizione interni dei quattro maggiori elementi, potendo effettuare un miglior confronto visivo. In secondo luogo, la rappresentazione dell'estremo inferiore e superiore dell'intervallo di confidenza, calcolato al 95%, permette di capire se e quali differenze fra le frequenze assolute siano significative, in modo da attribuire una validità statistica ai parametri estratti dalla distribuzione campionaria. Nel dettaglio, il numero di volte in cui la categoria "*dance*" è comparsa all'interno delle dinamiche della discussione su Twitter appare significativamente maggiore rispetto a quello relativo a "*rave*", ma non rispetto a "*jazz*" e "*rock*".

8 Frequenza assoluta dei generi nella discussione a tema musicale

Un approfondimento grafico sulla presenza delle principali categorie all'interno delle dinamiche della discussione virtuale è offerto mediante realizzazione di un *wordcloud* costruito con *Python*. Grazie alla semplicità di questa visualizzazione, l'osservatore può percepire visivamente la differenza fra le varie grandezze, ordinate a comporre la forma di una nota musicale.

9 Critiche alla visualizzazione e margini di miglioramento

Le critiche poste dal campione di fruitori durante l'osservazione e la valutazione del grafico *barchart* hanno riguardato maggiormente la mancanza di un adeguato livello di intuitività della visualizzazione, dovuta principalmente alla presenza di molti elementi grafici in uno spazio ridotto. Nonostante ciò, questa problematica non pare aver mai penalizzato gravemente la fruizione del grafico e la comprensione finale dell'oggetto descritto dall'analisi. Infatti, durante l'esecuzione di un'attività interattiva - che consisteva nel chiedere di selezionare alcuni generi in base a precise caratteristiche - gli utenti coinvolti sono riusciti in ogni caso a soddisfare correttamente le richieste.

Altro problema riscontrato è l'inserimento degli intervalli di confidenza nel grafico totale: i non addetti al settore hanno trovato di difficile interpretazione la lettura delle barre rosse a indicare i limiti superiori e inferiori. Pertanto si sono limitati a ignorarne la presenza considerandoli più elementi di disturbo che informativi.

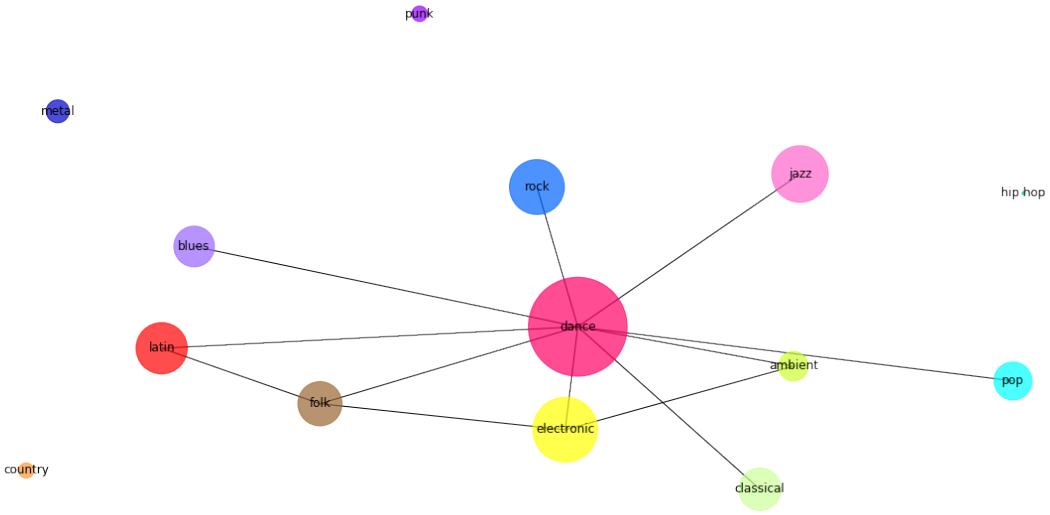
Per quanto riguarda il *wordcloud*, le uniche critiche sono state mosse da persone di età superiore ai 50 anni d'età: tale grafico può risultare una novità e può anche risultare poco intuitivo. Tuttavia, spiegato il funzionamento, questo è stato particolarmente apprezzato da tutti i fruitori, soprattutto a causa della sua semplicità.

10 Risultati dei questionari psicometrici

Parte V
Risultati e conclusioni

Il programma ha funzionato correttamente per tutto il breve periodo in cui la macchina virtuale è rimasta online, permettendo la raccolta di 2595 *tweets*: nonostante il numero non sia particolarmente alto, è sufficiente per eseguire alcune analisi quali l'identificazione delle comunità.

Nonostante i tweet rappresentino la musica di cui si parla sul social e non quella ascoltata, si nota subito che non esistono comunità musicali particolarmente chiuse, se non in quattro casi. Queste interagiscono col resto della piattaforma parlando principalmente della loro musica (presumibilmente preferita) e non di altri generi. Queste quattro comunità sono le minori per dimensioni: probabilmente il loro isolamento è dovuto più al basso numero di *tweet* raccolti che ad una tendenza delle persone che la compongono. Al contrario, argomento di discussione comune alla maggior parte degli utenti è la musica *dance*, tipica delle discoteche e di altri luoghi di ritrovo. Il risultato mostra che gli utenti preferiscono discutere delle esperienze in comune piuttosto che del proprio genere preferito, rendendo difficile quindi una clusterizzazione tramite il contenuto dei loro messaggi.



Bibliografia e sitografia

[1] Apache Software Foundation, cur. *Welcome to Apache TEZ*. 2019. URL: http://tez.apache.org/.

[2] Neo4J Inc., cur. *B.2. Use the Import tool*. v3.5. URL: https://neo4j.com/docs/operations-manual/current/tutorial/import-tool/.

[3] Neo4J Inc., cur. *Chapter 1. Introduction*. v3.5. URL: https://neo4j.com/docs/operations-manual/current/introduction/#enterprise-edition.

[4] Roopesh Shenoy. *What is Apache Tez?* Apr. 2014. URL: https://www.infoq.com/articles/apache-tez-saha-murthy/.

[5] Basho Technologies, cur. *Tutorial - Riak Python Client*. v1.5. URL: https://riak-python-client.readthedocs.io/en/1.5-stable/tutorial.html.