

Twitter Polarization and Vaccines

Guida Operativa

Armanini Justin – 830103 – j.armanini@campus.unimib.it

Caiffa Emanuele – 872515 – e.caiffa@campus.unimib.it

Palomba Eleonora – 876479 – e.palomba4@campus.unimib.it

Zayeva Nataliya – 867981 – n.zayeva@campus.unimib.it

Contenuti

Architettura	1
1. Twitter API	1
2. Apache Kafka	1
3. TigerGraph	2
4. MongoDB	4
Procedura	5
Ordine degli script	5
Riferimenti	6

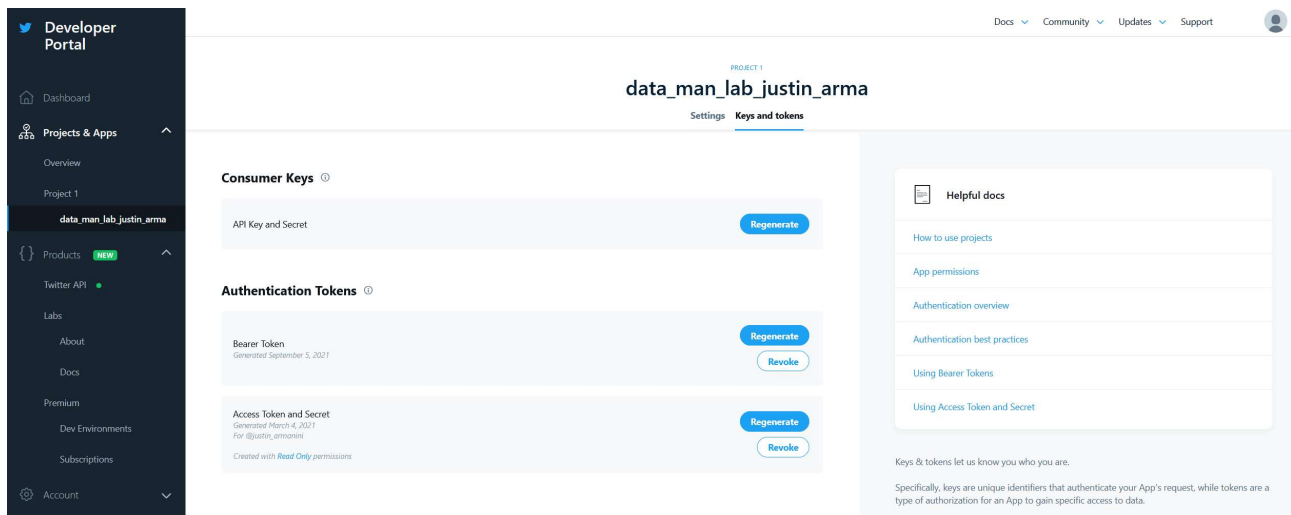
Si riportano i passi per configurare e mettere in funzione l'intera architettura implementata, analisi incluse.

Architettura

1. Twitter API

Per poter acquisire i dati Twitter tramite API è necessario innanzitutto creare un account Twitter Developer. Una volta fatto ciò, è possibile accedere al “Developer Portal”:

<https://developer.twitter.com/en/portal/dashboard>.



Attraverso la Web App è possibile dunque creare un nuovo progetto e generare le chiavi necessarie per poter inviare le richieste API tramite codice. In particolare serviranno:

- Access Token
- Access Token Secret
- Consumer Key
- Consumer Secret Key

Si presti attenzione a memorizzare questi dati (sono tutte stringhe alfanumeriche), poiché non saranno più visualizzabili, motivo per cui, in caso di smarrimento, sarà necessario rigenerarne di nuovi.

Una volta memorizzate queste informazioni è possibile quindi sfruttare le API. Nel progetto è stata utilizzata la libreria di Python “tweepy” [1] per poter scaricare i tweet in streaming.

2. Apache Kafka

Per installare e utilizzare *Apache Kafka* [2] è stata consultata la guida seguente: <https://towardsdatascience.com/running-zookeeper-kafka-on-windows-10-14fc70dcc771>.

Si noti che l'esecuzione di *Apache Kafka* richiede che sulla macchina stessa sia installato e sia in esecuzione un server *Apache Zookeeper* [3] per poter gestire l'intera configurazione di Kafka.

Per mettere in produzione il sistema è necessario quindi scaricare sia *Apache Zookeeper* che *Apache Kafka*. Una volta fatto ciò ed avere impostato tutte le variabili d'ambiente come riportato nella guida, bisogna avviare prima il server *Zookeeper*. Si apre un terminale e dalla cartella `<path zookeeper>/bin/` è sufficiente lanciare il comando:

zkserver

```
.\kafka-server-start.bat C:\Apache\kafka_2.12-2.7.0\config\server.properties
```

A questo punto il sistema Kafka è funzionante e può essere utilizzato.

3. TigerGraph

Nell'architettura proposta viene utilizzato *TigerGraph Cloud* [4]. Per poterlo utilizzare è richiesta la registrazione su tigergraph.com. Una volta completata la registrazione è possibile accedere ad una Dashboard che permette di configurare un'istanza di *TigerGraph* tramite interfaccia grafica:

Create Solution

1 Basic Settings 2 Instance Settings 3 Solution Settings 4 Confirmation

Select TigerGraph Version *

3.0.6
TG 3.1.1 is only available for single node for now. We will support cluster soon.

Select a Starter Kit *

Pick a Starter Kit with sample graph data schema, dataset, and queries (e.g. Fraud Detection, Recommendation Engine, Supply Chain Analysis, etc.).
Additional information including overview video at tigergraph.com/starterkits.

Anti-fraud Geospatial Analysis Graph Algorithms Healthcare
Knowledge Graph Machine Learning Recommendations
Temporal Analysis

Blank v3.0.6

Cybersecurity Threat Detection - IT v3.0.6

Enterprise Knowledge Graph (Crunchbase) v3.0.6

Fraud and Money Laundering Detection (Fin. Services) v3.0.6

Graph Analytics - Community Detection Algorithms v3.0.6

Healthcare - Referral networks, Hub(Pagerank) & Community Detection v3.0.6

In-Database Machine Learning for Big Data Entity Resolution v3.0.6

Network and IT Resource Optimization v3.0.6

Social Network Analysis v3.0.6

COVID-19 Analysis v3.0.6

Data Lineage v3.0.6

Entity Resolution (MDM) v3.0.6

GSQ 101 v3.0.6

Graph Analytics - Shortest Path Algorithms v3.0.6

Healthcare Graph (Drug Interaction/FAERS) v3.0.6

Low-Rank Approximation Machine Learning v3.0.6

Recommendation Engine (Movie Recommendation) v3.0.6

Supply Chain Analysis v3.0.6

Customer 360 - Attribution and Engagement Graph v3.0.6

Enterprise Knowledge Graph (Corporate) v3.0.6

Financial Services (Payments) - Fraud Detection v3.0.6

Graph Analytics - Centrality Algorithms v3.0.6

Graph Convolutional Networks v3.0.6

In-Database Machine Learning Recommendation v3.0.6

Machine Learning and Real-time Fraud Detection v3.0.6

Recommendation Engine 2.0 (Hyper-Personalized Marketing) v3.0.6

Si seleziona la versione, e successivamente un “progetto”. Ne esistono di diversi già realizzati con dati e query precaricati. Nel caso di interesse si seleziona un progetto vuoto.

Create Solution

1 Basic Settings 2 Instance Settings 3 Solution Settings 4 Confirmation

Select a Platform *

Choose a cloud platform.

aws Azure Google Cloud

Select an Instance Type *

Choose a virtual machine based on your data size and compute performance needs.

TG-Free
4 vCPU, 7.5GB Memory
\$0.00 / hour / instance
(See Details)

TG-C4.M16
4 vCPU, 16GB Memory
\$1.20 / hour / instance

TG-C8.M15
8 vCPU, 16GB Memory
\$1.80 / hour / instance

TG-C8.M32
8 vCPU, 32GB Memory
\$2.20 / hour / instance

TG-C16.M64
16 vCPU, 64GB Memory
\$4.10 / hour / instance

TG-C16.M122
16 vCPU, 122GB Memory
\$7.00 / hour / instance

TG-C32.M244
32 vCPU, 244GB Memory
\$12.00 / hour / instance

TG-C64.M488
64 vCPU, 488GB Memory
\$21.00 / hour / instance

TG-C96.M768
96 vCPU, 768GB Memory
\$30.00 / hour / instance

Select Region *

Choose region of the instance according to your networking requirements.

Ireland Tokyo Frankfurt London N. Virginia N. Cal

Disk Size *

Enter your disk size.

50 GB 128 GB 256 GB 512 GB 1024 GB 2048 GB GB \$0.03 / hour / instance

Partition Factor *

Number of data splits. See [FAQ](#) for more details.

2
E.g. 1 copy of data distributed into 3 partitions means partition factor 3.

Replication Factor *

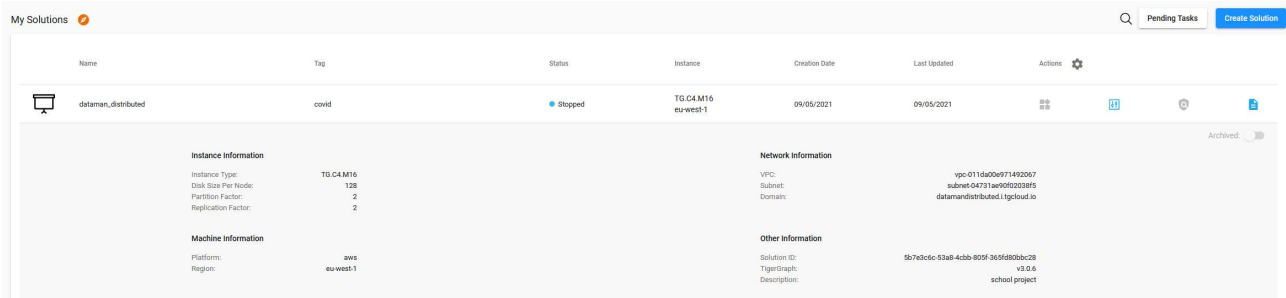
Number of data copies. See [FAQ](#) for more details.

2
E.g. 2 data copies means replication factor 2.

Trattandosi di una soluzione in cloud è possibile scegliere il tipo di piattaforma (AWS, Azure o Google Cloud) e il tipo di macchina che ospiterà l'istanza di TigerGraph. A seconda delle esigenze, si sceglie quindi l'alternativa più adatta. Infine si definiscono:

- **Partition Factor** cioè il numero di nodi in cui la soluzione verrà distribuita; in questo modo l'architettura dati realizza la scalabilità orizzontale dei dati;
- **Replication Factor**: cioè il numero di repliche dei dati; in questo modo l'architettura fornisce un servizio di fault-tolerance consentendo la continuità del servizio anche nel caso in cui uno dei nodi in cui i dati sono memorizzati dovesse cadere o interrompersi.

Ne consegue che il numero di nodi totali di cui si compone l'architettura è pari a *Partition Factor* \times *Replication Factor*. Nel progetto, considerato il volume dei dati a disposizione, si è deciso per una soluzione con sia *Partition* che *Replication Factor* pari a 2, per un totale di 4 nodi utilizzati. Successivamente, ai punti 3 e 4 della fase di creazione della soluzione viene richiesto di impostare il nome (per potervi accedere via web), password, tag e una breve descrizione. Infine, una volta ricapitolate le impostazioni, è possibile dare conferma e avviare il sistema.



Name	Tag	Status	Instance	Creation Date	Last Updated	Actions
dataman_distributed	covid	Stopped	TG.C4.M16 eu-west-1	09/05/2021	09/05/2021	Stop, Start, Restart, Delete, Edit Tags, Edit IAM Role, Edit VPC, Edit Subnet, Edit Domain, Edit DNS, Edit Route, Edit Security Group, Edit Elastic IP, Edit EBS Volume, Edit EBS Snapshot, Edit IAM Policy, Edit IAM User, Edit IAM Group, Edit IAM Role, Edit IAM Policy, Edit IAM User, Edit IAM Group, Edit IAM Role

Instance Information	Network Information
Instance Type: TG.C4.M16 Disk Size Per Node: 128 Partition Factor: 2 Replication Factor: 2	VPC: vpc-011da09e971492067 Subnet: subnet-04731ae9002038f5 Domain: datamandistributed1.tgcloud.io

Machine Information	Other Information
Platform: aws Region: eu-west-1	Solution ID: sb7e3dc0-53a8-4cbb-809f-3636f6b0bc28 TigerGraph: v3.0.6 Description: school project

Per poter interagire con il DBMS è possibile sia utilizzare la Web App *GraphStudio* che le API attraverso il package *pyTigerGraph* di Python.

4. MongoDB

MongoDB [5] è stato utilizzato non per memorizzare i dati veri e propri oggetto di analisi, ma per memorizzare i risultati delle fasi di elaborazione intermedi, utili per le analisi finali. In particolare è stato utilizzato per memorizzare le metriche necessarie per stabilire l'influenza settimanale degli utenti e le interazioni tra utenti influenti e non, come già spiegato nella Sezione 4 del paper del progetto.

Per poter implementare l'architettura dati, anch'essa distribuita (Sharding è il termine tecnico utilizzato) è necessario avere installato *Docker* e *MongoDB*.

In particolare è stato utilizzato il progetto: <https://github.com/kayne87/mongodb-sharding-docker>. Bisogna quindi clonare il repository.

Si apre il terminale nella cartella del repository e si lancia il comando:

```
docker-compose up -d
```

In questo modo viene inizializzata un'istanza di *MongoDB* con la seguente configurazione:

- due *shard cluster* in replica set, ciascuno con tre nodi *mongod*;
- un *config server* in replica set con tre nodi;
- un *router* per gestire il tutto.

Poi si accede alla shell di mongo con:

```
docker exec -it mongos1 /bin/bash
mongo
```

Infine all'interno della shell di mongo si abilita lo sharding dei dati per il database *dataman_project* e si creano le collezioni necessarie:

```
use dataman_project
```

```
sh.enableSharding("dataman_project")

# Collection metriche
db.metriche.createIndex({_id: "hashed"})
sh.shardCollection("dataman_project.metriche", {"_id" : "hashed"})
db.metriche.getShardDistribution()

# Collection links
db.links.createIndex({_id: "hashed"})
sh.shardCollection("dataman_project.links", {"_id" : "hashed"})
db.links.getShardDistribution()
```

Ora anche mongoDB è pronto per poter essere utilizzato.

Procedura

Si descrive l'ordine delle operazioni da effettuare per poter riprodurre il progetto.

- Configurare tutti i software come descritto nella Sezione **Architettura**

Ordine degli script

Tutti gli script sono stati scritti utilizzando Python 3.8, per cui per eseguirli è sufficiente avere installata una versione uguale o superiore sul proprio PC e lanciare `python3 <nome_script>.py`. In alternativa si può utilizzare un qualsiasi IDE, come Visual Studio Code.

1. Le API per interagire con TigerGraph sono state “wrappate” secondo il design pattern “*Adapter Pattern*” creando una libreria custom chiamata ***tigergraphAPI.py***. Al suo interno è stata definita la classe *tigergraph_connection*, che tra i vari metodi implementati espone i metodi *load_queries()* e *install_queries()*. Il primo serve per caricare le query necessarie per le analisi sul DBMS, il secondo per “installarle”. L’installazione di una query è un concetto proprio di TigerGraph che consiste nell’ottimizzare la query in oggetto, permettendo così di migliorarne le prestazioni. Lo svantaggio è che una qualsiasi modifica della query richiede la sua reinstallazione. Questi due metodi sono quindi da eseguire soltanto la prima volta che ci si connette al DBMS, quando questo è ancora vuoto e non ha alcuna query in memoria;
2. Si esegue ***kafka_producer.py*** per raccogliere i tweet e memorizzarli nella coda del topic *tweets*;
3. Si esegue ***kafka_consumer.py*** per leggere i tweet dal topic *tweets* e inserire il grafo che rappresenta il tweet appena letto, gli utenti coinvolti, ed eventualmente il tweet a cui quello corrente sta rispondendo (si faccia riferimento alla modellazione dei dati descritta nella Sezione 3 del paper). Nel caso in cui uno degli utenti coinvolti sia già presente, il che significa che esiste già un nodo nel grafo memorizzato fino a quel punto che lo rappresenta, il grafo

inserito non sarà una componente connessa, ma verrà “attaccato” a quello già memorizzato fino a quel punto;

4. Una volta raccolti tutti i tweet per il periodo di interesse si esegue lo script ***influcent_classifier.py***. Per ciascuna settimana esso interroga il database TigerGraph, calcola le metriche necessarie per stabilire l’influenza degli utenti, le struttura in un DataFrame pandas a cui viene aggiunta la colonna *week* (per permettere di filtrare per settimana in fase di interrogazione) e salva tali DataFrame (uno per ogni settimana) nella collection *metriche* di mongoDB;
5. Si esegue ***influcent_update_graph.py***: sempre per ciascuna settimana, legge dalla collection *metriche* i documenti riferiti a quella settimana, ricostruisce il DataFrame creato al punto 4 e legge la *classe* di ciascun utente (ovvero se è influente o meno). Aggiorna i nodi *User* del grafo coerentemente con la *classe*. Legge dal grafo tutte le relazioni che esistono tra un utente influente e uno non influente che ha retwittato almeno un tweet del primo. Si ottiene così un DataFrame con 2 colonne, utente influente e utente non influente che ha retwittato il primo, ed esattamente come fatto al punto 4 aggiunge la colonna *week* per le interrogazioni successive, salvando infine nella collection *links*.
6. Per ciascuna settimana si esegue un notebook dedicato ***<n>_community_detection.ipynb*** che legge dalla collection *links* i documenti della settimana di interesse ricostruendo il DataFrame descritto al punto 5, crea il grafo di proiezione, esegue l’algoritmo di community detection, e infine calcola la polarità associata a ciascun utente. Data la polarità di tutti gli utenti, tramite Kernel Density Estimation (KDE) si ottiene una stima della Densità di probabilità della polarizzazione.
Si noti che per la prima settimana è stato esportato il notebook jupyter in formato HTML, di modo da spiegare nel dettaglio tutte le fasi di analisi.

Riferimenti

- [1] «tweepy,» [Online]. Available: <https://docs.tweepy.org/en/stable/>.
- [2] «Apache Kafka,» [Online]. Available: <https://kafka.apache.org/>.
- [3] «Apache Zookeeper,» [Online]. Available: <https://zookeeper.apache.org>.
- [4] «TigerGraph Cloud,» [Online]. Available: <https://docs.tigergraph.com/cloud>.
- [5] «MongoDB,» [Online]. Available: <https://docs.mongodb.com/>.