Twitter Analytics Documentation

Release 0.1

Deepak Saini, Abhishek Gupta

Jun 08, 2018

CONTENTS:

1 Introduction to twitter analytics system **3**

2 Read data from twitter streaming API **5**

3 Ingesting data into Neo4j **7**

4 Ingesting data into MongoDB **9**
- 4.1 Basics First . . . 9
- 4.2 Ingestion Rates . . . 9
- 4.3 Code Documentation . . . 10

5 Neo4j: API to generate cypher queries **13**
- 5.1 Template of a general query . . . 13
- 5.2 Creating a custom query through dashboard API : Behind the scenes . . . 14
- 5.3 Code Documentation . . . 14

6 Generating queries in mongoDB **17**

7 Composing multiple queries : DAG **19**

8 Generating alerts using flink and kafka **21**

9 Benchmarking the query answering **23**

10 Dashboard Website **25**

11 Indices and tables **27**

Python Module Index **29**

Index **31**

Twitter generates millions of tweets each day. A vast amount of information is hence available for different kinds of analyses. The end users of these analyses however, may or may not be technically proficient. This necessitates the need of a system that can absorb such large amounts of data and at the same time, provide an intuitive abstraction over this data. This abstraction should allow the end users to specify different kinds of analyses without going into the technicalities of the implementation.

In this demonstration, we introduce a system that tries to meet precisely the above needs. Running on streaming data, the system provides an abstraction which allows the user to specify real time events in the stream, for which he wishes to be notified. Also, acting as a data-store for the tweet network, the system provides another abstraction which allows the user to formulate complex queries on this historical data. We demonstrate both of these abstractions using an example of each, on real world data.

Here we provide a comprehenstive documentaion of each component of the system along with a documentation of the code.

CHAPTER

ONE

INTRODUCTION TO TWITTER ANALYTICS SYSTEM

CHAPTER

TWO

READ DATA FROM TWITTER STREAMING API

CHAPTER

THREE

INGESTING DATA INTO NEO4J

CHAPTER

FOUR

INGESTING DATA INTO MONGODB

4.1 Basics First

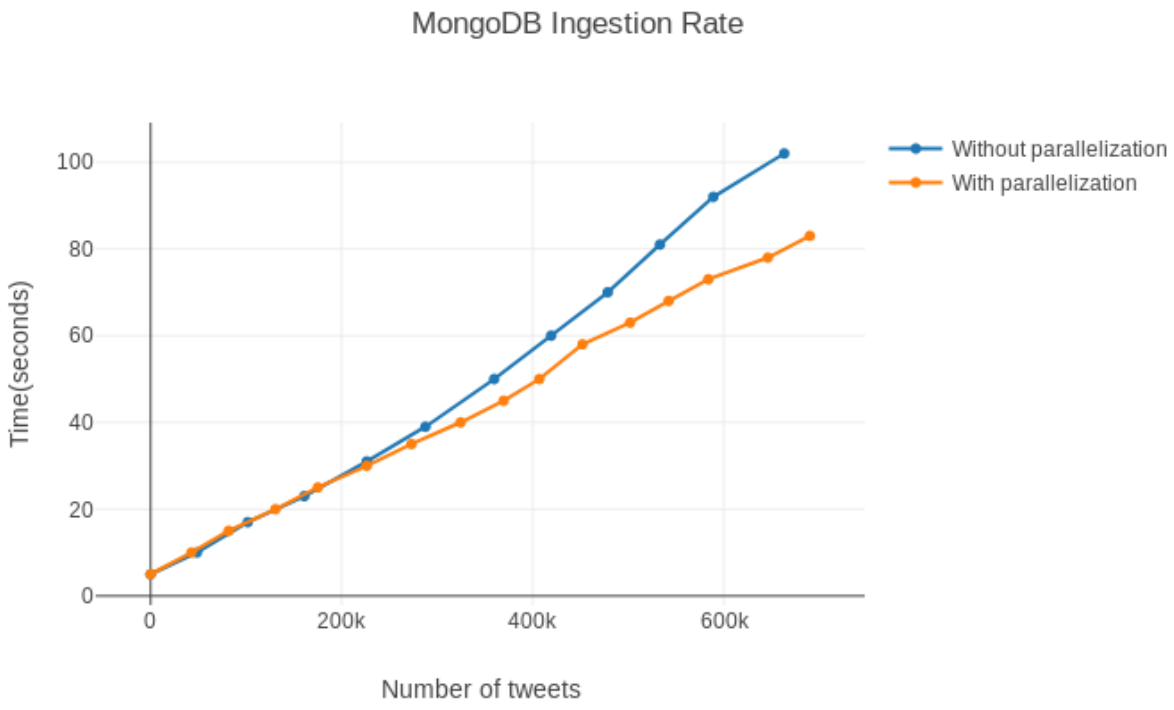
It has no network based data, simple data just sufficient for analytical queries. We are storing some basic statistics of the tweets in mongoDB for faster access. Presently we are storing :

- The number of times a hashtag, url, user mention has appeared in time intervals of 1 min.
- The basic sentiment associated with a hashtag, as count of number of positive and negative words

So how are we ingesting? We have two processes(most of the times running on different cores), with P1 keeping track of the statistics. After every min, the data is put into a pipe which is received by P2 and then parallely put into mongoDB.

4.2 Ingestion Rates

AS expected, the ingestion rate into mongoDB shilw overlapping writing into diska nd building the new batch is faster than without parallelization. This can be observed in this image:



4.3 Code Documentation

class ingest_raw.**Ingest**(*interval*)

Bases: object

aggregate()

exit()

insert_tweet(*tweet*)

update the in memory dictionaries

populate()

write to the mongoDB

worker(*q*)

class ingest_raw.**Timer**(*interval*, *function*, *args=None*, *kwargs=None*, *iterations=1*, *infinite=False*)

Bases: multiprocessing.context.Process

Calls a function after a specified number of seconds:

```
>>> t = Timer(30.0, f, args=None, kwargs=None)
>>> t.start()
>>> t.cancel() #stops the timer if it is still waiting
```

authkey

cancel()

Stop the timer if it hasn't already finished.

daemon
Return whether process is a daemon

exitcode
Return exit code of process or *None* if it has yet to stop

ident
Return identifier (PID) of process or *None* if it has yet to start

is_alive()
Return whether process is alive

join(*timeout=None*)
Wait until child process terminates

name

pid
Return identifier (PID) of process or *None* if it has yet to start

run()

sentinel
Return a file descriptor (Unix) or handle (Windows) suitable for waiting for process termination.

start()
Start child process

terminate()
Terminate process; sends SIGTERM signal or uses TerminateProcess()

ingest_raw.**calculate_sentiment**(*positive_words*, *negative_words*, *tweet_text*)

ingest_raw.**getDateFromTimestamp**(*timestamp*)

ingest_raw.**read_tweets**(*path*, *filename*)

ingest_raw.**threaded**(*fn*)

CHAPTER

FIVE

NEO4J: API TO GENERATE CYPHER QUERIES

Here we expalin the API to generate cypher queries for Neo4j.

5.1 Template of a general query

Any query can be thought of as a 2 step process -

- Extract the relevant sub-graph satisfying the query constraints (Eg. Users and their tweets that use a certain hashtag)
- Post-processing of this sub-graph to return desired result (Eg. Return "names" of such users, Return "number" of such users)

In a generic way, the 1st step can be constructed using AND,OR,NOT of multiple constraints. We now specify how each such constraint can be built.

We look at the network in an abstract in two dimensions.

- There are "Entities" (users and tweets) which have "Attributes" (like user has screen_name,follower_count etc. and tweet has hashtag,mentions etc.).
- The entities have "Relations" between them which have the only attribute as time/time-interval (Eg. Follows "relation" between 2 user "entities" has a time-interval associated).

So each constraint can be specified by specifying a pattern consisting of

- Two Entities and their Attributes
- Relation between the entities and its Attribute (which is the time constraint of this relation)

To make things clear we provide an example here. Suppose our query is - Find users who follow a user with id=1 and have also tweeted with a hashtag "h" between time t1 and t2. We first break this into AND of two constraints:

- User follows a user with id=1
- User has tweeted with a hashtag "h" between time t1 and t2.

We now specify the 1st constraint using our entity-attribute abstraction.

- Source entity - User, Attributes - None
- Destination entity - User, Attributes - id=1
- Relationship - Follows, Attributes - None

We now specify the 2nd constraint using our entity-attribute abstraction.

- Source entity - User, Attributes - None
- Destination entity - Tweet, Attributes - hashtag:"h"

- Relationship - Follows, Attributes - b/w t1,t2

The missing thing in this abstraction is that we have not taken into account that the source entity in both the constraints refers to the same User. To do so, we "name" each entity (like a variable). So we have:

- **Constraint 1:**
  - Source entity - u1:User, Attributes - None
  - Destination entity - u2:User, Attributes - id=1
  - Relationship - Follows, Attributes - None
- **Constraint 2:**
  - Source entity - u1:User, Attributes - None
  - Destination entity - u3:Tweet, Attributes - hashtag:"h"
  - Relationship - Follows, Attributes - b/w t1,t2

5.2 Creating a custom query through dashboard API : Behind the scenes

A user can follow the general template of a query as provided above to build a query. when a user provides the inputs to specify the query, the following steps are executed on the server:

- Cleanup and processing of the inputs provided by the user.
- The variables(User/Tweet) and the relations are stored in a database. These stored objects can be later used by the user.
- The query specified by the user is converted into a Cypher neo4j graph mining query.
- Connection is established with the neo4j server and the query is executed on the database.
- The results obtained are concatenated and are displayed.

5.3 Code Documentation

Here we provide a documentation of the code. Module to generate cypher code for inputs taken from user though dashboard API.

The *generate_queries* module contains the classes:

- *generate_queries.CreateQuery*

One can use the *generate_queries.CreateQuery.create_query()* to build a cypher query.

Example illustrating how to create a query which gives the userids and their tweet counts who have used a certian hashtag.

```
>>> actors=[("u","USER"),("t","TWEET"),("t1","TWEET")]
>>> attributes=[[],[("hashtag","{hash}")],[]]
>>> relations=[("u","TWEETED","t","",""),("u","TWEETED","t1","","")]
>>> cq = CreateQuery();
>>> return_values="u.id,count(t1)"
>>> ret_dict = cq.create_query(actors,attributes,relations,return_values)
>>> pprint(ret_dict,width=150)
```

Example of a query which uses time indexing in a relationship:

```
>>> actors = [('x', 'USER'), ('u1', 'USER')] + [('t1', 'TWEET'), ('t2', 'TWEET')]
>>> attributes = [[('id', '12')], [('id', '24')]]+[[('hashtag', 'BLUERISING')], [(
↪'retweet_of', 't1'), ('has_mention', 'u1')]]
>>> relations = [('x', 'FOLLOWS', 'u1', '', ''), ('x', 'TWEETED', 't2', '24', '48')]
```

class generate_queries.**CreateQuery**

Bases: object

Class containing functions to generate query.

conditional_create(*entity*)

Condionally provide the attributes of the node if not already created, else directly use the name of the variable create earlier. If already create, pass empty list of properties.

Parameters **entity** – the entity which to check and create

Returns the code for the node as neo4j node enclosed in ()

create_query(*actors*, *attributes*, *relations*, *return_values*)

Takes a list of attributes and relationships between them and return a cypher code as string. For the format of the lists see the examples.

Parameters

- **actors** – the variable names, types of the attributes
- **attributes** – the properties of the actors
- **relations** – the relations between the entities along with time index for the relations
- **return_values** – a direct string containing the return directive.

Returns index of the bond in the molecule

Note: Here we are expecting that if user has not specified the times on the dashboard, then we pass epmty string. If you store some other default in dashboard database then change this accordingly.

Note: The return _values is directly used as a string in the cypher query, so the user can use AS and other similar cypher directives while specifying the query.

Todo: Support to compose queries using OR. For example, currently compostion of relationships or attribute properties like all tweets(t) which are retweets of t1 or quoted t2, is not supported. Use cypher union for this.

---

generate_node(*var*, *type*, *props*)

Helper function for *generate_queries.CreateQuery.conditional_create()*

Parameters

- **var** – the variable name of the entity
- **type** – the type of the entity. Observe we pass type as :USER and NOT as USER
- **props** – the properties of the entity.

Returns the code for the node as neo4j node enclosed in ()

CHAPTER

SIX

GENERATING QUERIES IN MONGODB

CHAPTER

SEVEN

COMPOSING MULTIPLE QUERIES : DAG

CHAPTER

EIGHT

GENERATING ALERTS USING FLINK AND KAFKA

CHAPTER

NINE

BENCHMARKING THE QUERY ANSWERING

CHAPTER

TEN

DASHBOARD WEBSITE

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

generate_queries, 14

i

ingest_raw, 10

INDEX

A

aggregate() (ingest_raw.Ingest method), 10
authkey (ingest_raw.Timer attribute), 10

C

calculate_sentiment() (in module ingest_raw), 11
cancel() (ingest_raw.Timer method), 10
conditional_create() (generate_queries.CreateQuery method), 15
create_query() (generate_queries.CreateQuery method), 15
CreateQuery (class in generate_queries), 15

D

daemon (ingest_raw.Timer attribute), 10

E

exit() (ingest_raw.Ingest method), 10
exitcode (ingest_raw.Timer attribute), 11

G

generate_node() (generate_queries.CreateQuery method), 15
generate_queries (module), 14
getDateFromTimestamp() (in module ingest_raw), 11

I

ident (ingest_raw.Timer attribute), 11
Ingest (class in ingest_raw), 10
ingest_raw (module), 10
insert_tweet() (ingest_raw.Ingest method), 10
is_alive() (ingest_raw.Timer method), 11

J

join() (ingest_raw.Timer method), 11

N

name (ingest_raw.Timer attribute), 11

P

pid (ingest_raw.Timer attribute), 11
populate() (ingest_raw.Ingest method), 10

R

read_tweets() (in module ingest_raw), 11
run() (ingest_raw.Timer method), 11

S

sentinel (ingest_raw.Timer attribute), 11
start() (ingest_raw.Timer method), 11

T

terminate() (ingest_raw.Timer method), 11
threaded() (in module ingest_raw), 11
Timer (class in ingest_raw), 10

W

worker() (ingest_raw.Ingest method), 10