Semantic Data Management

BDMA JOINT PROJECT

Public Opinion Mining for Investments

Team MindMiner

Akash Malhotra, Ali Arous, Haonan Jin, Yalei Li, Yuhsuan Chen

Git: https://github.com/akashjorss/SDM_BDM_Joint_Project

Business Introduction

With the advent of the internet, market research for investment cannot rely solely on conventional tools anymore1. Internet technologies like social media have added another dimension to this process, i.e., the collective opinion of the people. With studies showing a strong correlation between movements of financial markets and public sentiments2, there is a need for a tool which can accurately analyse this trend. MindMiner aims to provide such a tool for investors to perform public opinion mining about companies on the relevant data from social media and sources on the web. We want to provide an integrated solution which provides near real-time analytics, sentiment analysis on a piece of data and the data as a whole, and historical analysis, in a user-friendly way. We propose to take advantage of graph analytics and stream processing to offer clients more insights about their investment.

1. Graph-based solution [M1]

Our major goal is to provide the sentiment analysis based on public opinions (e.g. tweets) for investors, and we provide two levels of analysis, the tweet/comment level and the hashtag/topic level. While tweet level sentiment analysis results indeed provide very useful information to our customers in real time, the overall or general sentiment tendency towards topics is more appealing and can provide more informative insights to our customers.

This happens when a certain investor is interested in unveiling the trending topics related to a company of his interest, its most sold products, its newly achieved milestone in number of subscribers etc. However, this is more beneficial when it comes side by side with the general opinion about each topic. These topics translate in Social Media terminology to Hashtags.

In order to achieve sentiment analysis on the hashtags, we will first materialize the different hashtags in a database such that they reside linked to each other, and to the tweets they originated from. An algorithm proposed by Wang et al[3] is then run over the structure to compute the individual sentiment for each hashtag based on the collective sentiments of other related hashtags as well as its tweets. Multiple network algorithms will then be executed over the graph (i.e. centrality, label propagation, community detection) to provide more insights to customers based on the hashtag sentiments as a whole.

2. Purpose statement [M1]

The connected information within the graph database can ingest different data sources and provide market insights to the customers. Graph database natively embraces relationships and is able to store, process, and query connections efficiently, while other databases compute relationships at query time through expensive JOIN operations.

Accessing nodes and relationships in a native graph database is an efficient, constant-time operation and allows us to quickly traverse millions of connections per second per core[4]. Independent of the total size of the dataset, graph databases excel at managing highly-connected data and complex queries. It enables us to explore the neighboring data around those initial starting points - collecting and aggregating information from millions of nodes and relationships - and leaving any data outside the search perimeter untouched.

Therefore, we would like to provide a graph-based solution of the sentiment analysis for our customers, which ensures query efficiency, development flexibility, and insight enrichment. Being able to mine the public opinions and generate valuable metadata that describe them provides an opportunity to understand the general sentiment around the business operations in a democratized way. By democratizing the subjectivities, we can make an objective analysis of subjective content, giving us the ability to better understand trends around companies that our customers can use to make better investment decisions.

3. Property graph & Neo4j [M2]

When deciding the graph tools, property graph is chosen to be the most appropriate graph family for us. The major reason is the wide range of proven network analytics that come out of the box and are enabled in the property graph. Currently, we are aiming to solve problems of a single dimension (i.e. public opinion), and the analytical needs overweigh the semantics within the data. Various types of graphs can be stored in property graphs, like undirected and weighted, while knowledge graphs focus on storing rows of triples. In addition, the ability of maintaining attributes on the edges within the property graph will offer us more development efficiency in case of network analytics. For instance, weighted sentiment score can help in detecting the most influencing user, where the weights can be obtained within the edges. Temporal information (e.g. tweet time) may be also needed for our future functionalities.

It is also essential for us to adopt the suitable tools in order to run the algorithm natively over a graph structure, and to eventually achieve the highest throughput possible. Among different alternatives available for big data analytics over graphs (e.g. GraphX, GraphFrames, Giraph), we opted for Neo4j graph database to ensure high application and database availability:

- Fault Tolerance: re-routes activity to other available servers when a clustered server fails.
- Multi-Data-Center Support: allows masters and read-replicas in a server cluster to reside in different data centers—while maintaining high performance across all data centers.
- Hot Backups: allow the user to make full and incremental backups across data centers while your app is running to maximize uptime and ease administration.
- High Availability for Distributed Workloads: enables the user to separate read requests from other operations by physically and logically locating the graph as close to users and apps as possible.

Neo4j also has a “Neo4j Spark Connector” which uses the binary Bolt protocol to transfer data from and to a Neo4j server. It offers Spark-2.0 APIs for RDD, DataFrame, GraphX and GraphFrames, so we give ourselves a room to choose how to use and process our Neo4j graph data in Apache Spark in the future to offer big data analytics services on the graph.

Finally, Neo4j comes with a Graph Data Science library that incorporates the predictive power of relationships and network structures in existing data and combines a native graph analytics workspace and database with graph algorithms and visualization to take analytics over graphs to a new scale.

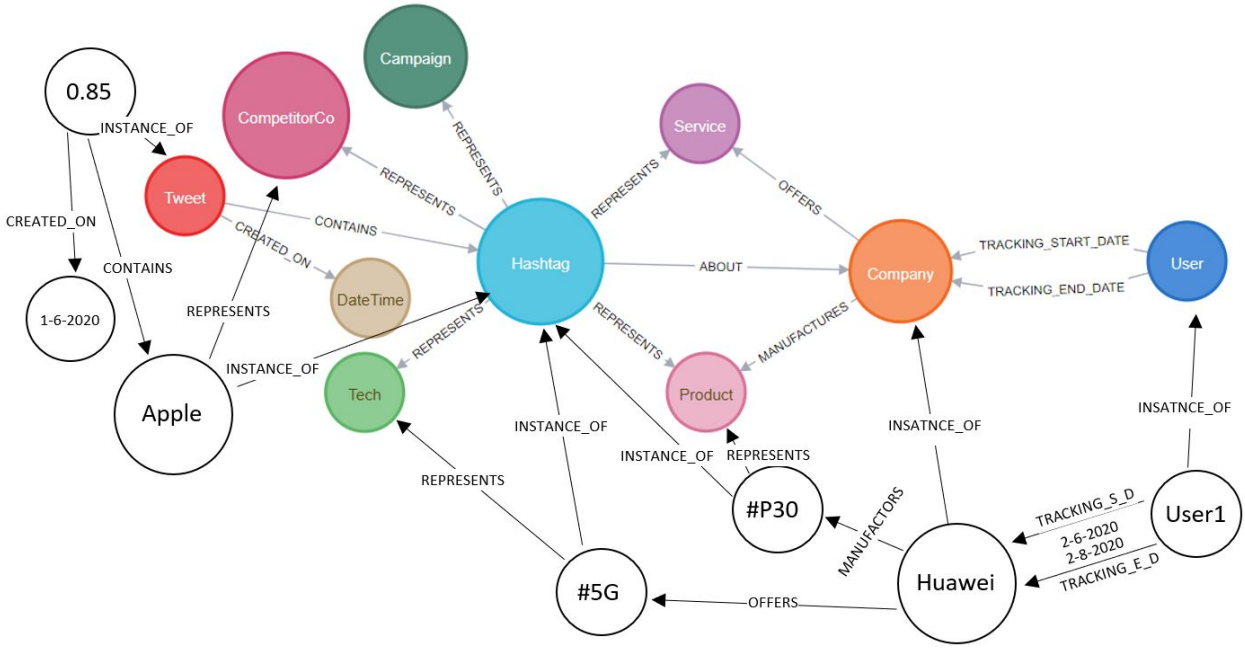
4. Graph Design [M3]

Meta Data Description, Graph Schema & Instances:

For our overall graph schema, we aim at categorizing the different hashtags into the types they represent. For example, a given hashtag can represent a product that the company manufactures, a service offered by the company, a competitor company, or even a social media campaign pro or against the company’s brand. In this sense we use the relationship

REPRESENTS to link instances of Hashtag to their proper category. We would like to draw the attention here that the depicted schema is autogenerated from Neo4j browser, so having several relationships between Hashtag nodes and the different categories is a result of different Hashtag instances each of them being linked to one of the categories.

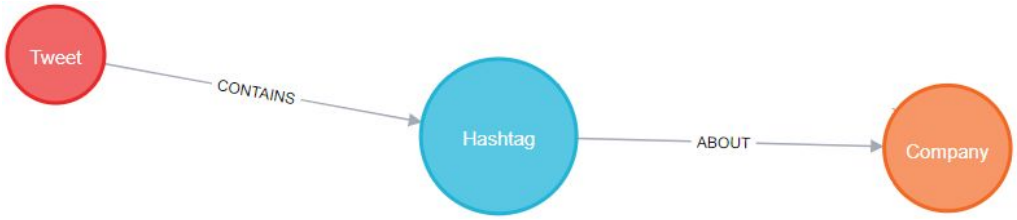
This can be achieved either by using some third party NLP library or by relying on lookup dictionaries where store the different terms under one category, and then simply do string matching using regular expressions.



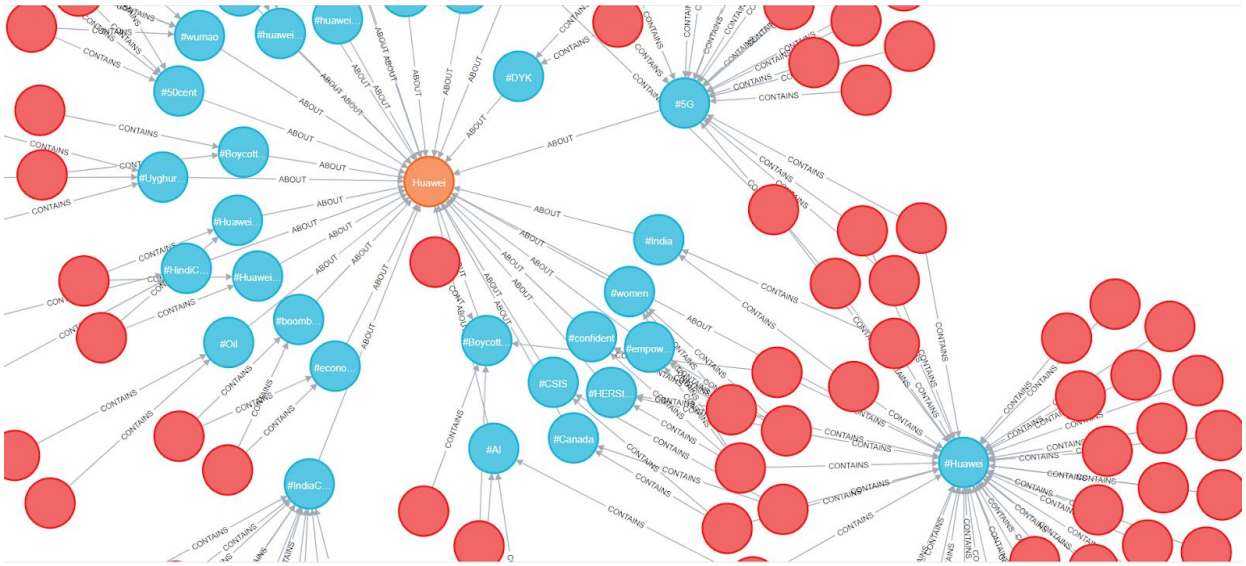
To make the above diagram more readable, we omitted the ABOUT relationship from Hashtags instances to the Company instance, and sufficed by referring to it at the schema level.

The projected operational Graph Schema:

For the sake of the prototype, we narrow down the overall graph schema into a minimal one focusing only on the hashtag-tweet-hashtag relations, in what is formally known in the literature as “Hashtag co-occurrence graph”. We then run our analytics over this graph. We show below the Neo4j generated schema of Hashtag co-occurrence graph:



Below shows a snapshot of our instances, extracted from 3000 tweets talking about “Huawei”.



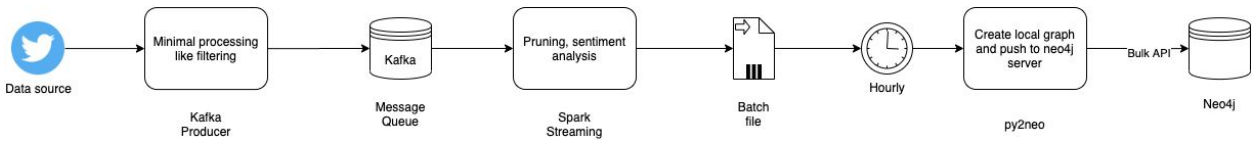
5. Data Flow [M4]

In the production ready system, the data will come from multiple sources which can be characterised into two categories:

API based: These sources have a well defined schema and data can be retrieved using REST API usually in the form of json schema. This data goes into the Kafka brokers and then can be reformatted, pruned and processed by spark streaming which automatically retrieves this data from the brokers.

Web scraped: If the source doesn't have an API, then we write spiders to scrape it. Using XPath, we can retrieve the relevant data from the source and then send it to the Kafka Broker, from where spark streaming can retrieve it and process it.

After Spark-streaming is done with processing the data, it loads it into a batch file. This batch file is used to create a local graph using py2neo library in python, which is then pushed into the remote graph using bulk api on an hourly basis using a workflow scheduler. For our prototype, we only load data from the twitter API and we do not use a scheduler, but we load data from batch files manually.



6. Graph exploiting [M5]

In Twitter, hashtags are a community-driven convention for adding additional context and metadata to tweets. They are created organically by Twitter users as a way to categorize messages and to highlight topics. The extensive use of hashtags makes Twitter more expressive and welcomed by people. A recent study[3] measured on a dataset with around 0.6 million randomly selected tweets and found that around 14.6% of them have at least one hashtags in each. When only considering the subjective tweets (tweets with positive/negative sentiment expressions), this number increases to 27.5%. The statistics show a great potential for sentiment analysis with hashtags in Twitter.

To prepare the sentiment analysis on the hashtags, we first materialize different hashtags in the database such that they reside linked to each other, and to the tweets they originated from. We follow the academic approach[3] to calculate final sentiment labels for hashtags related to a certain company of our customer's interest by incorporating two types of information:

1. The sentiment polarity of tweets containing the hashtag
2. The hashtags co-occurrence relationship

Data preparation and analysis

The data preparation and analysis flow is as follows:

1. Compute the sentiment on the tweet level using a pre-trained ML model.
2. Filter out objective tweets
3. Normalize the tweet level sentiments into range [0,1]
4. Normalize hashtags to lower case
5. Filter out hashtags occurring less than a given threshold τ
6. For each hashtag compute the probability of it getting each sentiment label individually (i.e. prob_1 and prob_0).
7. Give each hashtag the initial sentiment label of the highest probability.
8. Project the Hashtag-Tweet-Hashtag graph to Hashtag-Hashtag graph
9. Calculate the weight of the edge between each pair of hashtags in the projected graph based on the number of times they co-appeared in tweets.
10. Apply Label propagation algorithm to propagate the initial sentiment labels through the weighted co-occurrence graph.
11. Repeat 10 until convergence

Steps 2, 3, 4, 5 are optional. They can be switched on/off and compared against each other.

We start the analysis flow at step 6, by initializing the hashtags with preliminary sentiment labels based on their neighborhood's tweets (i.e. tweets from which the hashtag originated).

The sentiment label $\hat{y_i}$ is decided by maximizing the following function:

$$\hat{y}_i \leftarrow \underset{y \in \{pos, neg\}}{\arg\max} \frac{\sum_{\tau \in \mathcal{T}_i} \Pr_{y_i}(\tau)}{\sum_{\tau \in \mathcal{T}_i} \sum_{y} \Pr_{y}(\tau)} \quad (1)$$

If we denote the $b_i(y_i) \leftarrow \frac{\sum_{\tau \in \mathcal{T}_i} \Pr_{y_i}(\tau)}{\sum_{\tau \in \mathcal{T}_i} \sum_{y} \Pr_{y}(\tau)}$ possibility that a hashtag h_i is labeled with assignment y_i , We need to calculate the possibility that h_i is labelled positive $b_i(pos)$, and the possibility that it is labelled negative $b_i(neg)$.

In order to deal with the tweet level sentiments as probabilities, we need to first scale their values from the range [-1, 1] into [0,1] using the following formula:

$$z_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

Then to compute $b_i(pos)$ we sum the positive sentiment probabilities of the tweets in h_i neighborhood, and divide it by the sum of the probabilities of those tweets being either positive or negative [which is equal to: $b_i(pos) + b_i(neg)$] . We do the same approach for $b_i(neg)$ but this time we take (1-tweet_sentiment) as the probability of being negative. (since it is the complementary event of being positive).

After computing $b_i(pos)$, $b_i(neg)$ we can easily maximize the formula in (1) by comparing the two values and taking the label that corresponds to the higher of them. By now we have calculated the initial sentiment labels for hashtags using the aggregated sentiment of their neighboring tweets. Next step is propagating those labels through the hashtag-tweet-hashtag network, and updating them iteratively until we reach a stationary state.
To do this we need to create a projection of our graph, by calling the following graph data science procedure, which does cypher projection over the the specified parameters:

```
1. CALL gds.graph.create.cypher(
2.     'g',
3.     'MATCH (h:Hashtag) RETURN id(h) AS id, h.sent_label AS sent_label',
4.         'MATCH (h1:Hashtag)-[:CONTAINS]-(t)-[:CONTAINS]-(h2:Hashtag) RETURN id(h1) AS source, id(h2) AS target, COUNT(t) AS weight'
5. )
```

In the previous script, we project the hashtag node along with its sentiment label, and we also project the relationship: Hashtag-Tweet-Hashtag as a one edge in the projected graph, with a property named “weight”. This weight will represent the number of tweets that exist along the way between two given hashtags (say, h1 and h2). So the more a pair of hashtags co-occur in a

single tweet, the stronger the relation between them, and hence the more effective will be to propagate the label form one of them to the other.

Now we apply the Label Propagation algorithm that does the rest of the work by propagating the labels of initial sentiments through the weighted hashtag co-occurrence graph, doing that in iterations, until a stationary state is achieved.

```
1. CALL gds.labelPropagation.stream('g', {seedProperty: 'sent_label', relationshipWeightProperty:
   'weight' })
2. YIELD nodeId, communityId AS Community
3. RETURN gds.util.asNode(nodeId).name AS Name, Community
4. ORDER BY Community, Name
```

After running the algorithm over a dataset of 3000 tweets, we got the following statistics:
ranIterations: 3, **didConverge:** True, **createMillis:** 0**, computeMillis:** 8

After the results we got shows that, out of 94 hashtags, 21 changed their initial sentiment label. 1 hashtag changed from -1 to 0, 2 hashtags changed from 0 to 1, and 18 changed from 0 to 1.

The full Cypher implementation of the discussed analysis can be access at our github repo:

https://github.com/akashjorss/SDM_BDM_Joint_Project

After the sentiment value sentiment score is assigned for each tweet, we can then develop the function that we offer to our user. In our product, we offer company analysis, industry analysis and statistics analysis.

Use Case - product function

1. Present average hashtag sentiment score for one company.

The characteristic of graphs can quickly achieve this function. We only need one jump from the hashtag node to tweet and calculate the average sentiment score of each tweet by hashtag. When the client only wishes to be interested in the positive hashtag, we can also modify the query and return the score larger than zero. In contrast, the score smaller than zero will represent a negative. We will be able to offer recommendations of most welcoming companies based on the average sentiment score rank over the companies.

2. Present the sentiment score during a specific time, (e.g. From May 01 to today).

This function allows the user to see the fluctuation of the sentiment trend of a particular company. We model DateTime in an independent node for users to easily access the data from a certain date.

3. Present the top 3 hashtags for each company - keyword analysis.

This function can provide users with an overview of the trending keywords for a company. For example, the company Apple has around 70 hashtags, but we obtain the most appear 3 hashtags are [AppleStore, AppleWatch, Mac], and for Nvidia is [Art, CGI, MachineLearning].

4. Present trending company in the industry.

This function can provide users with the rank of the top discussed company by calculating the number of tweets and the rank for each company to give them more ideas when making investment decisions.

5. Present most used Hashtag for each company or popular companies

This can be achieved using the PageRank algorithm. By running this centrality algorithm over one or many companies, the popular hashtags can be recognized, and then we can return with average sentiment, company and number of tweets. It can assist our end users with investment-worthy insights of the market hotspots, and/or identify the trending topics of a specific company.

6. Present the sentiment score of each community

We run community detection algorithms, Louvain and Label Propagation to partition the related hashtags into communities and do sentiment analysis over them to get the score by community. Take Amazon as an example (query result shown below): we discover that the community that relates to Alexa has a negative public attitude, while Amazon business community such as eCommerce, marketing or Amazonstore has a positive public attitude, which is beneficial to the investor to choose what area they are going to invest for Amazon product.

community	Hashtags	numberOfTweets	avgSentiment	Company
1054	["#AskAlexa"]	93	-0.9375	"amazon"
1297	["#amazon", "#iartg", "#histfic", "#Rome"]	41	0.3538839615668884	"amazon"
1341	["#ONEUS_FOR_THE_THRONE", "#원어스_1등_가자_우"]	32	0.175	"google"
1279	["#AB6IX"]	24	0.0596153846153846	"google"
1249	["#free", "#deals", "#freebies", "#giveaway", "#discount", "#offers"]	22	0.23696969696969697	"amazon"
186	["#LISA", "#LALISA", "#BLACKPINK", "#KPOP", "#YG"]	20	0.0	"google"
1076	["#google", "#Section230"]	20	0.022954545454545457	"google"
1351	["#ecommerce", "#business", "#store", "#tips", "#marketing", "#seo", "#wordpress"]	16	0.8854166666666666	"amazon"
394	["#AppLocker", "#security", "#windows", "#whitelisting", "#microsoft"]	14	0.2865936147186147	"microsoft"
1280	["#TWICE", "#SANA", "#트와이스", "#사나"]	12	0.0	"apple"
1478	["#newrelease"]	12	-0.025868055555555533	"amazon"
1141	["#Art", "#MachineLearning", "#CGI", "#ComputerGraphics"]	11	0.4375	"nvidia"
1131	["#WWDC20", "#appleevent", "#WWDC2020", "#WWDC", "#WWDCScholars", "#Apple"]	10	0.0	"apple"
1317	["#YuWensheng", "#HumanRights", "#China"]	9	0.0	"apple"
1236	["#NintendoSwitch", "#AnimalCrossing", "#ACNH", "#ketchuptheduck"]	8	0.12500000000000003	"apple"

So far, in the prototype, we develop the above preliminary functions with the help of Neo4j build-in algorithm. More delicate services can be achieved by taking advantage of the characteristics of graph-based tools to provide more insights.

7. Proof of Concept [M7]

Tools chosen:

Requirement	Tool chosen	Alternative	Reason
Language	Python3	Java	Good for quick prototyping and to test feasibility. Java is bulky and takes time. We will migrate to java for production.
Message queue	Kafka	SNS, other cloud solutions	Open source and cheaper, easy to test in local dev environment
Stream processor	Spark streaming	Flink, Kinesis, NiFi	Fault tolerant, Good for testing locally as opposed to cloud solution, low learning curve compared to Flink. We don't need strict real time processing, so a micro-batch approach is acceptable.
Graph DBMS	Neo4j	OrientDB, MongoDB, etc	Native graph storage, good for traversal queries, consistent, fabric server for distributed processing, good UI, good integration with spark for production systems. Good for collaborative prototyping with Neo4j Sandbox.
Graph Processing	Neo4j DS Lib	Giraph, GraphX, etc.	Easy integration with neo4j DBMS. Distributed query processing using fabric server.
Sentiment analysis	Textblob	Google nlp, stanford corenlp	Fast due to simplicity, lightweight library, cloud solutions too slow for streaming
Libraries	Pyspark, py2neo, pykafka	N/A	Python libraries to do high level development using tools used.

PoC setting:

Macbook air (1.6 GHz Processor, Intel Core i5, 8 GB DDR3 RAM, run in a pip virtual environment). We use neo4j sandbox for analytics.

End to end example:

1. Start zookeeper & kafka server, then make a partition called “huawei " with default replication factor.

2. Start a python script that listens to retrieves all the tweets from twitter containing the keyword “huawei”. It then sends this tweet as a message to the kafka partition with the same name using producer api

3. Start Spark streaming context with a micro-batch interval of 1 second and create a direct stream using pyspark.kafka_utils which listens to the kafka partition called “huawei”.

4. Spark streaming does the following processes to the tweet:

    a. Load the tweet to a json format using json.loads(tweet)

    b. Removes all the data not required by neo4j. (Only need tweet_id, text, hashtags, date, time, retweet_count)

    c. Filters the tweets which have at least one hashtag.

    d. Performs sentiment analysis on tweet text using textblob library add this as a sentiment field in the json document. Also add the company field to the json document, with value “huawei”.

    e. Writes the data of the microbatch to a batch file(json) huawei_tweets.json.

5. Read the data from huawei_tweets.json, create a local graph with the given schema using py2neo and load it into the neo4j graph via bolt. Delete the batch file.

6. Run the label propagation, louvain and pagerank using neo4j browser and calculate sentiment of the company using the method discussed above.

7. Visualize the results in the desired format.

References

1. Independent Investment Research Tools | TD Ameritrade. (2020). Retrieved from https://www.tdameritrade.com/investment-research.page

2. Dickinson, B., & Hu, W. (2015). Sentiment analysis of investor opinions on twitter. *Social Networking, 4*(03), 62.

3. Wang, X., Wei, F., Liu, X., Zhou, M., & Zhang, M. (2011, October). Topic sentiment analysis in twitter: a graph-based hashtag sentiment classification approach. *In Proceedings of the 20th ACM international conference on Information and knowledge management* (pp. 1031-1040).

4. Bastani, K. (2020). Deep Learning Sentiment Analysis for Movie Reviews using Neo4j. Retrieved from https://www.kennybastani.com/2014/09/deep-learning-sentiment-analysis-for.html