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 Summary

With the advent of the internet, market research for investment cannot rely solely on conventional tools[1] anymore. Internet technologies like social media have added another dimension to this process, ie, the collective opinion of the people. With studies[2] showing a strong correlation between movements of financial markets and public sentiments, 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 from 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 on 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] 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.

1. Introduction

Data Sources

We plan to perform sentiment analysis on Fortune 500 companies, to begin with, where the full company list will be generated from the official website for automatic topic partitioning in Kafka. Public opinions towards companies will be collected from multiple sources with the development of the business. We will start with twitter and amazon reviews. The preliminary semantic analysis will be performed on each message, and the results will be stored in a graph database for queries and more complex algorithms. Users can trace an interested company for real-time updates about how the market responds to the company's business strategy, operation, new product launch etc.

For the Twitter data, we use Twitter API and connect to Apache Kafka to directly ingest the data in a scalable manner. The velocity of tweet streaming will be nearly real-time, where we can offer the most up-to-date market information to customers. According to the research, it is expected to reach an average of 6,000 tweets per hour for a popular company, which leads to a monthly increase of around 50 GB per company.

For the Amazon reviews, we perform a scraping program over the major products review URLs of each company to extract the required details from it (i.e. Review of Title, Helpful Count, User Review and Date of Review). The velocity of Amazon review will be daily, given the update frequency of most Amazon products. Regarding academic publication, the volume is expected to be around 1,000 reviews for a single product. On average, a company has five main products. That translates to 5000 reviews and 100 MB of data every day. Together with the tweet data, a monthly incremental in data storage will be around 3 GB per company.

Since we gather unstructured data sources from various platforms, the heterogeneity is recognized by us. Though challenging to integrate, we believe the heterogeneity of data can offer more market insights to our customers. We deploy graph databases to better serve the variety in sources due to their schemaless nature. To gain flexibility in data modelling, we must maintain a Data Lake to store the data first, and process later. It can help us maintain cost-effective big data storage, and facilitate the R&D where users may demand different functionalities from us.

Data Storage & Management

The data model is defined given the consideration of the future query usage and intermediate analytical process. Real-time streaming data routes are established within Kafka to reliably pass data to Spark Streaming by running on a cluster of servers. The Kafka cluster stores and categorizes streams of tweets in Topics (i.e.

companies), while these records consist of key, value, and timestamp. The Kafka cluster always retains all published records, without consideration of their consumption. However, in our proposed framework there is no need to store imported data in the database, and the whole process will take place in memory to avoid access latency of hard disks. Spark is a Big Data processing framework built to offer speed and sophisticated analytics, where it runs streaming sentiment analysis as a series of very small, deterministic batch jobs[3]. These batches may consist of Tweets in a short time as low as half a second with a latency of about one second.

Ingested data will be stored in the data lake first. Here, we use MongoDB, a NoSQL database to accommodate the different data sources and formats. It is essential for the database to be scalable and distributed (sharding) in order to host both the large volumes of historic and streaming data at the same time. Moreover, the support for a NoSQL query is essential to retrieve company-based data from the text. Original information will be stored with the partition of companies, and different data sources are stored in their respective kafka partition to decouple the later operations.

Then, ingested data will be processed for sentiment analysis. We use a pretrained model like Google NLP API to predict sentiment value for each message in a constant time. We feed data to Elastic stack so that end users can access stream analytics in real time. Persistent sentiment results with selective raw information (e.g. tweet id, tweet text, hashtags) are stored in the graph database to facilitate future query.

Data Analysis

Data cleaning is conducted within Kafka when ingesting the data. Certain parts are filtered out of the data and then processed to Spark, such as text, tweet id, hashtags, etc. Within Spark streaming, we will apply sentiment analysis with a pre-trained model to get the sentiment result of each message. It is regarded as the data preparation for the graph algorithm. Meanwhile, basic real time analytics can be directly visualized to our customers by Kibana, like the number of positive tweets about a specific topic in the past 5 minutes.

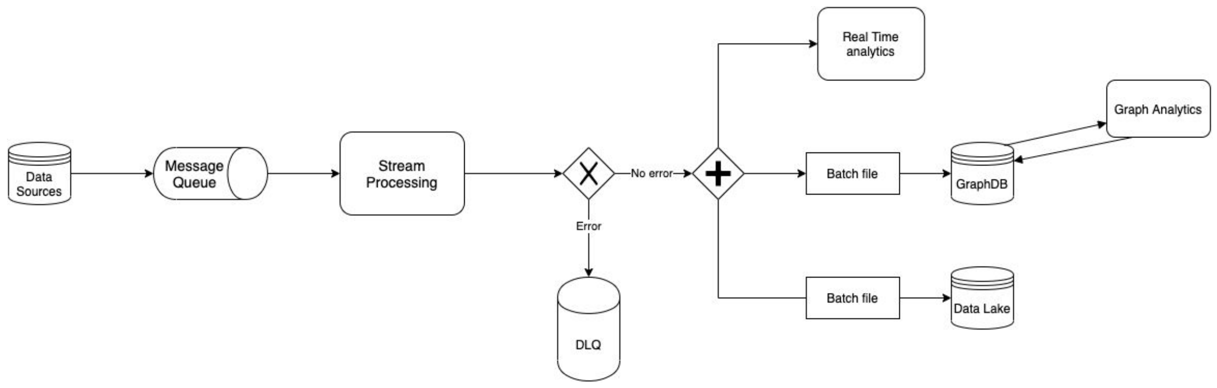
In order to achieve sentiment analysis on the hashtags within graph databases, we need to first materialize the different hashtags in a database such that they reside linked to each other, and to the tweets they originated from. An algorithm proposed by Wang et al[7] is then run over the structure to compute the individual sentiment for each hashtag based on the collective sentiments of other related hashtags as well as its tweets. At the end, various graph-based algorithms will be conducted. For instance, centrality algorithm (i.e. PageRank) will provide our users with investment-worthy insights of the market hotspots, and/or identify the trending topics

[3] M. Z. Mosharaf Chowdhury and T. Das, "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in NSDI'12 Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. San Jose, *CA: USENIX Association Berkeley,* Apr. 2012.

of a specific company. Label propagation and Louvain algorithms are used to partition the related hashtags into communities and provide users with the market hotspots based on the community sentiment score ranking.

In addition, data partition by companies will facilitate sequential reads when querying/analyzing the market response of a single company. Also, parallel processing can be realized on each data partition when running an algorithm.

2. Functional architecture



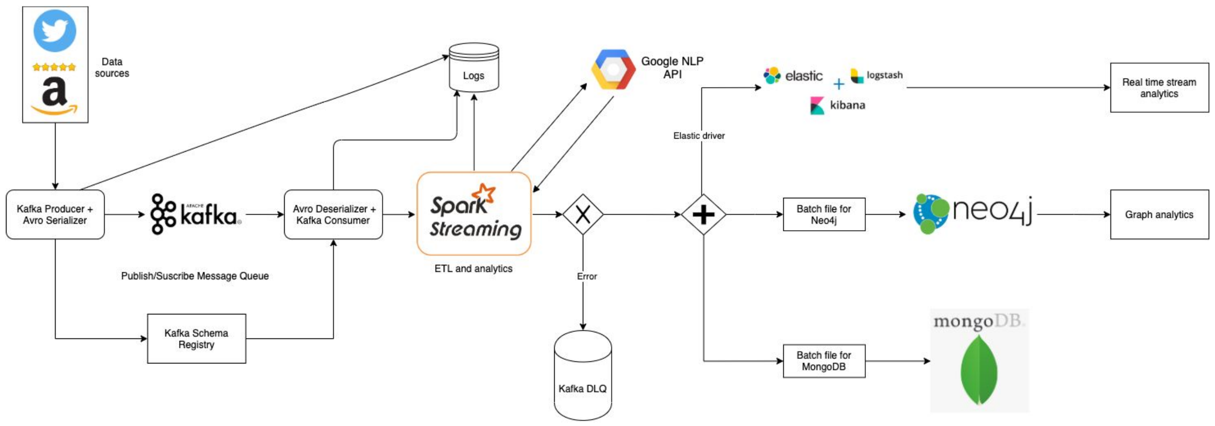
The architecture is developed to provide both real-time and historical analysis. A message is an atomic unit in our data. In our architecture, we have the data sources (i.e. real-time tweets from Twitter API & scrapped Amazon reviews), from which the messages will be ingested by Apache Kafka. Kafka is a distributed streaming platform that uses publish-subscribe messaging and is developed to be a distributed, partitioned, replicated service. Our framework uses this message brokering system. To balance the incoming load, Topics are defined by companies and each of these Topics is split into multiple partitions, each storing one or more of those partitions with ability to accept multiple formats. Multi-subscriber is allowed in Kafka and may have zero, one, or many consumers who may access the data written to it. Partitions are sequences of records which are ordered, immutable, and may be continually appended to a commit log. This architecture allows Kafka's performance to be constant with respect to data size. Message queue is used to decouple the stream processing engine from the data source.

Data processing takes place using Spark distributed execution engine. In general, Spark Streaming receives live input data streams and processes each record individually once they arrive. Additional sentiment analysis algorithm which is built on Spark engine will be used to perform basic data processing. Spark will generate the final stream of results to storage. In this project, Spark is connected to Kafka as a direct consumer client instead of using Receivers. The direct approach ensures Exactly Once processing of the Kafka data stream messages. Full end-to-end

Exactly Once processing may be achieved provided that Spark's output processing is implemented as Exactly-Once[4].

If there is some error in processing of a message, it is pushed into a dead letter queue, otherwise it is loaded into a NoSQL DB, a graph db and into a real time visualisation tool parallely. The result of all three can be accessed from the serving layer. The visualisation is used to provide preliminary statistics in real-time, while more complicated algorithms will be used in the graph database to offer more market insights in a reasonable latency (i.e. few minutes). Data stored in the NoSQL DB data lake may be used later for historical data analysis.

3. Tool selection



Message Queue

To facilitate the data incoming and ease the job for data sources, we take advantage of the message queuing system to persist messages in the message queue, from where consumers can actively pull the message from the subscribed queue. With the business development, we may need to deal with hundreds of them. Thus, we need to deploy a queue cluster to process different data sources. Such a model can also ensure scalability, multi-node fault tolerance, data backup, etc. There exist many products in the market to help us handle the previously mentioned problems, such as Amazon Kinesis and Apache Kafka (Kafka). After considerable research, we decided to deploy Kafka in our project given its higher flexibility and potentially the expense (Kafka is open-sourced).

As a distributed streaming platform, Kafka is run as a cluster on one or more servers that can span multiple datacenters[5]. Kafka cluster stores the messages in a topic with keys, values and timestamps, and offers APIs to perform different roles. In this proposal, Kafka acts as a real-time distributed message queue cluster performing real-time message production and consumption. Avro serialisation is used with

[4] Yadranjiaghdam, B., Yasrobi, S., & Tabrizi, N. (2017, June). Developing a real-time data analytics framework for twitter streaming data. In *2017 IEEE International Congress on Big Data (BigData Congress)* (pp. 329-336). IEEE.
[5]Apache Kafka. (2020). Retrieved from https://kafka.apache.org/intro

schema registry to send and receive data from a Kafka broker because it can compress the messages and facilitate speedy delivery of messages. We also deploy the Spark Streaming real-time computing framework to consume the data in Kafka in real time and then perform operations. We plan to take the direct Kafka integration with Spark Streaming, where no receivers are set. Instead, it periodically queries the latest offsets from the corresponding partitions under Kafka's topics (company based), and then processes the data inside each micro-batch according to the offset range, which Spark reads certain data by calling Kafka's simple consumer API (low-level api). Several advantages can be achieved by this approach[6], such as simplified parallelism, improved efficiency, and exactly-one-semantics (EOS).

Stream Processing

Spark is deployed here due to its core distributed execution engine. External libraries powered by the engine allow us to perform complex operations. Sentiment analysis here is often iterative, and the caching ability of Spark can help us greatly speed up these iterative tasks. Sophisticated algorithms can be realized with the high-level functions like map, reduce, join, and window.

There are many possible candidates like Flink, Storm and Spark. Storm can solve only one type of problem i.e Stream processing. Here, we need a generalized solution for future potential problems like iterative processing and batch processing. We decided to go with Spark Streaming for a general purpose computation engine. Its market maturity, ease of use and python compatibility provide us more efficiency to integrate with the Hadoop ecosystem.

Spark Streaming abstracts the live stream by dividing it into batches (or micro-batches) of a predefined interval and then treats each batch of data as an array of Resilient Distributed Datasets (RDDs). Then we can process these RDDs using the operations like map, reduce, reduceByKey, join and window. Compared to Spark Streaming, other stream processing frameworks process the data streams per each event rather than as a micro-batch. With a micro-batch approach, we can use other Spark libraries (like Core, Machine Learning etc) with Spark Streaming API in the same application.

Just like Spark, spark streaming provides fault tolerance with its RDD lineage graph. It uses many worker nodes for parallel processing and is infinitely scalable. In future, we plan to use cloud services like AWS, which offer fully managed elastic compute framework for Spark where worker nodes can be added automatically on demand.

Graph DBMS

Besides the basic real-time sentiment statistics, we provide two other levels of sentiment analysis based on graph databases: the tweet/comment level and the

[6]Iozzia, G. (2019). *Hands-On Deep Learning with Apache Spark: Build and deploy distributed deep learning applications on Apache Spark* (1st ed., pp. 88-89). Packt Publishing Ltd, 2019.

hashtag/topic level. These overall or general sentiment tendencies towards topics are more appealing and can provide more informative insights to our customers.

Within graphDB, we can take advantage of the wide range of proven network analytics that come out of the box (algorithms as mentioned in section 1). To perform the analytical tasks, we need an efficient Graph Database 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 such that: 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 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. In our project, we utilize the centrality, page rank, and Louvin algorithms to generate more insights.

Data Lake

There are many possible candidates for storing our heterogeneous and historical data, like Hadoop data lake. Although Hadoop pioneered the "schema on read" idea of data lake and provided a cost-effective framework to store large amounts of data, there is not an actual way to turn the data into knowledge. Additionally, deciphering the data is often tested to be difficult for Hadoop[7]. However, MongoDB data lake provides a more specific concept, similar to "external table" in RDBMS. We are able to query the data using the familiar MongoDB syntax without actually loading them

[7] Harrison, G. (2020). The Advantages of MongoDB's Atlas Data Lake. Retrieved from https://www.dbta.com/Columns/MongoDB-Matters/The-Advantages-of-MongoDBs-Atlas-Data-Lake-134829.aspx

into MongoDB first. Besides, the collection of data can be directly stored within a cloud object in Amazon’s S3 service, which can considerably save storage cost.

MongoDB is a schemaless database system that stores documents in BSON format, which means we do not need to define the type of columns before inserting our data. We definitely will have more data sources in the future, most of which will have different schemas. It offers flexibility and allows us to maintain polymorphic storage which scales up easily. Besides, MongoDB offers a wide range of index types and allows us to declare primary and secondary indexes on any fields in the document to support the intricate access patterns. MongoDB indexes are on-demand that can support evolving application requirements and queries to optimize the performance.

We have connected Hadoop with a MongoDB database to automate two tasks that are needed for supporting real-time data ingestion and data processing of large volumes of streaming tweets, as well as supporting NoSQL queries for retrieving specific data (e.g. market hotspots) from these tweets. The Hadoop–MongoDB connector is needed for running the market-performance query because tweets contain unstructured data (e.g. user generated content such as images and messages) integrated with data that may be well structured individually (e.g. user id, timestamps, geographical coordinates) but which jointly defines a large and evolving data schema.

Real time visualisation

To provide real time statistics to our customers, like the number of positive tweets about a certain topic in the past 5 minutes, we need a visualisation tool that can provide live dashboards.

From the big players in the market Tableau, Amazon Quicksight and Kibana, we choose the latter for the following reasons:

- Kibana is an open source platform, with high level easy to use APIs, so that deployment can be automated. It also supports visualisation language Vega.
- It offers a higher refresh rate than quicksight and comparable to Tableau, which is crucial to show real time analytics.
- Kibana is based on Elasticsearch and Logstash For handline real-time data visualization and is easier to operate on the AWS platform, which offers us more possibility to provide varieties of real-time data visualization in future.

We can use Kibana to search, view, and interact with data stored in the Elasticsearch index. Using various charts, tables, maps, etc. Kibana can easily display advanced data analysis and visualization.

4. Data flows

Data Ingestion

Appendix 1.1 shows the ingestion processes for our major data sources. As mentioned in Section 1 (Data source), we will perform opinion mining for the companies products from Amazon’s review pages. Data will be crawled from review URLs to extract all required details. The gathered text needs special attention in order to satisfy the required format. For example, <br/> tags have a special meaning to the browser (break read or next line), and we need to explicitly convert each <br/> tag to spaces or else the crawling result value will be damaged. In order to credit the reviews, we collect the amount of up-vote and down-vote as Helpful Count to assist the review accountability. The following is the list of items that we have extracted: Company, Product/Service, Review of Title, Helpful Count, User Review and Date of Review.

The data extracted need to be cleaned so that we get proper text review on which analysis can be performed. Cleaning of crawled data is done by removal of all special characters (such as: “:/.,’#$*^&-) in order to retrieve best results. After cleaning the crawled content, we generate a csv file, store it in the data lake, and the data analysis process will be ready next.

Appendix 1.2 shows the Twitter data ingestion flow. As stated in Section 1, we take advantage of the publish-subscribe architecture of Apache Kafka to facilitate the Twitter data ingestion. For the sake of load balancing and query efficiency, topics are defined by companies, where producers get the stream of data from the Twitter API, and publish data to the desired Topics. Each record published to a Topic is accessed by one or more consumers (e.g. data lake, or graphDB). The Kafka cluster always retains all published records, without consideration of their consumption. However, in this report, there is no need to store imported data in the database, and the whole process will take place in memory to avoid access latency of hard disks., where it may be in separate processes or on separate machines.

It also sends the Avro Schema of the data to the Schema Registry [8] by Confluent. Then a Kafka Consumer Program first deserializes avro and then sends the data to spark streaming for processing.

Data Processing
For Amazon review data (appendix 1.3), after the data cleaning, we will classify the review type: for service, feature, or product (classification added in data storage). A service review test is conducted where we test the occurrence of service words. If the service test is failed, product feature test is performed using POS (part of speech) tagging and ruled based extraction. If the review fails the feature test also, then the review is classified as a product review. A new final csv is generated with the classification and sentiment of the feature phrases. This csv is then loaded into

[8]Schema Management — Confluent Platform. (2020). Retrieved from https://docs.confluent.io/current/schema-registry/index.html

the graph database for creating the visualizations by querying data from the database.

For tweet data processing, Spark Streaming receives input data stream and processes microbatch (DStream, 1 second long) of tweet RDDs. We directly connect Spark with Kafka without the Receivers to ensure Exactly Once processing, and with write ahead logs to ensure fault tolerance. Within Spark, remove irrelevant data from the tweet, filter the tweets which have invalid data for our graph (e.g. no hashtags), do some window based aggregations with checkpoints to visualize real time analytics and do sentiment analysis (via Google NLP API) for each incoming message. After this, the data is written to the batch file and the window based aggregations result is sent to Elastic Cloud for visualization. A local graph is created using the data in this batch file periodically and then pushed to Neo4j via bulk api periodically.

5. Performance metrics

Throughput: We determined empirically (also supported by some market research) that on average for one popular company 1.5 MB of data is generated per minute by twitter. We plan to store data of fortune 500 companies. This requires the bandwidth of 12.5 MB per second, which is quite achievable by a network connection to a computing node. Our system can easily handle ingestion of this amount of data.

Scalability: As mentioned before, for one company, around 50 GB of data is generated from twitter. If we have around 5 more sources like amazon reviews, which , on average, generates 3 GB of data per company, then we have 15 GB data per company per month. In Kafka we store data upto two weeks old, since the relevancy of sentiment analysis doesn’t last for more than that. So overall, for 500 companies, we need to store 16.25 TB of data in Kafka. If each node can store 1 TB of data, to achieve a replication factor of 3, we need a cluster of around 49 nodes. However, in the production, we plan to deploy on AWS, and we can set an auto scaling group, which can add/remove nodes automatically depending on the traffic volume.

Aligning with the above calculations, the maximum memory spark streaming needs is 48 GB (for 1 hour windows). For this, 3 workers with 16 GB memory is enough. However, we will use AWS EMR cluster which can automatically add/remove workers if needed.

Quality metrics
Accuracy of Sentiment Analysis: We use Google cloud NLP for sentiment analysis, which can perform bulk sentiment analysis with latency of less than 1 second. Google is considered to be the leader in NLP, and based on our empirical evaluation, their NLP engine is quite accurate. To show stream analytics however,

we use a local Naive Bayes model which can perform sentiment analysis with milliseconds latency.
Timeliness: To achieve less latency in repeated tasks like loading data to neo4j and mongodb, we use bulk load api which can load data very quickly and the speed is only limited by the network bandwidth. Thus, this can keep up with the streaming ingestion.
Latency in spark: Spark 0.5 second microbatch latency is not an issue as doing strict real time processing is not a priority for us. For real time stream analytics, we use a window of 5 mins to an hour and 0.5 second is insignificant compared to that.
Fault tolerance: We create a direct stream of kafka with spark streaming with write-ahead logs and use checkpointing with window based aggregations to ensure fault tolerance.

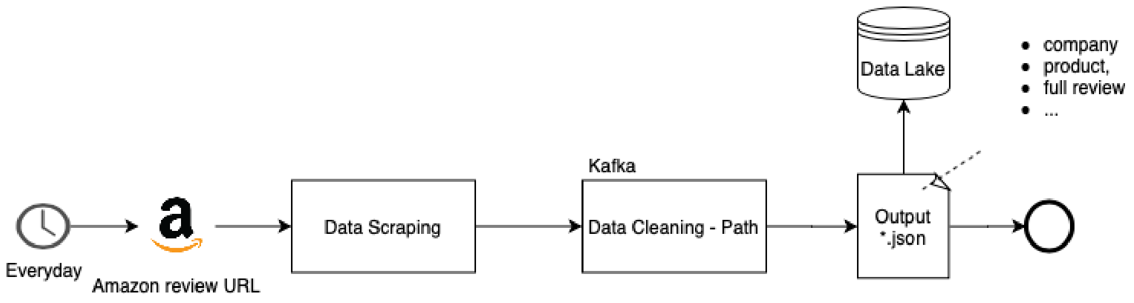
6. PoC description

To show the feasibility of our product, we run PoC on our local computer (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.
We use 10 Kafka partitions, 1 for each company we track with a replication factor of 3. We use following tools to perform various tasks:

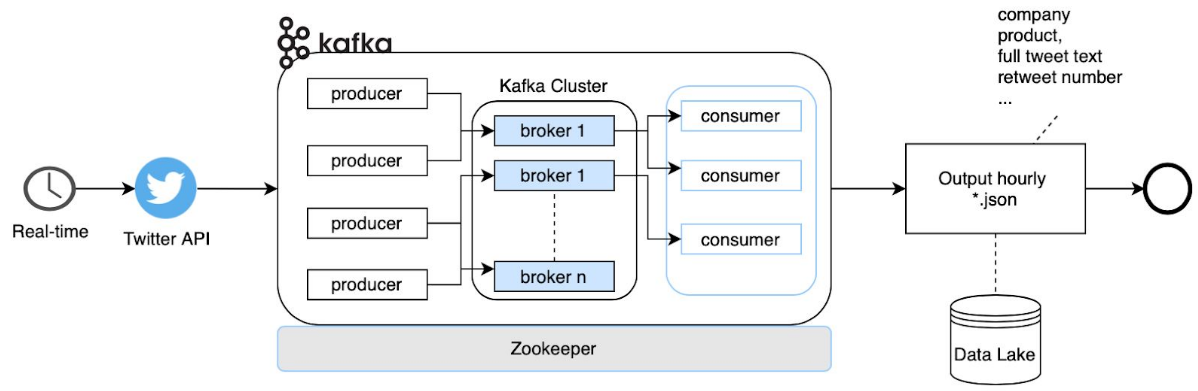
Tools	**Usage**
Python	Our development language
tweepy	Library for twitter ingestion. We track and ingest tweets of 10 companies using this.
pykafka	library to interact with kafka
pyspark	library to interact with spark streaming
py2neo	Neo4j python driver
pymongo	Mongodb python driver
elasticsearch	Elasticsearch python driver
textBlob	Library for sentiment analysis

Appendix 1: data flow figures

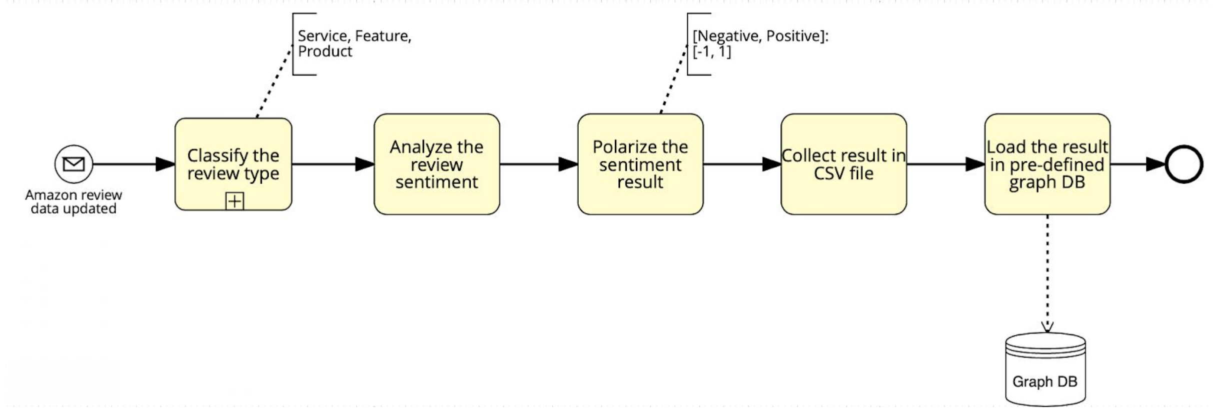
1. Amazon review data ingestion



2. Twitter real-time data ingestion

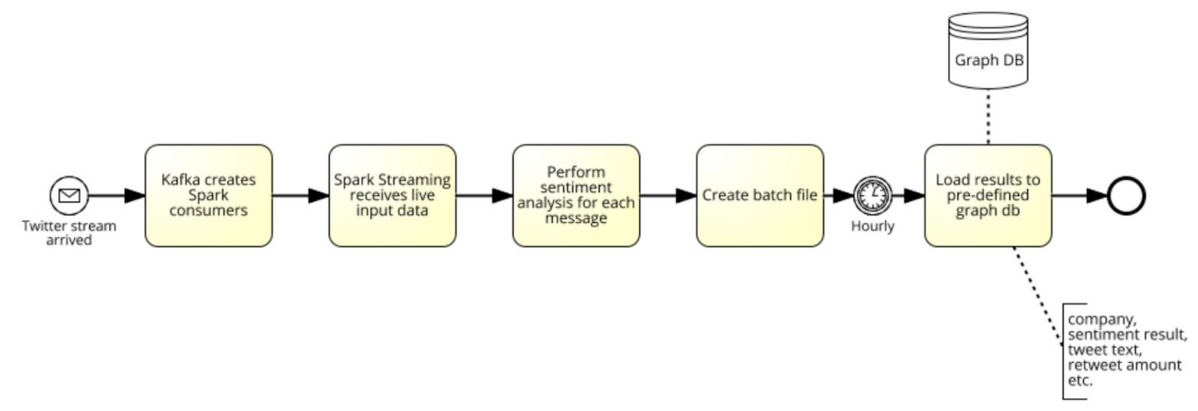


3. Amazon review data processing



Service, Feature, Product
[Negative, Positive]: [-1, 1]
Amazon review data updated
Classify the review type
Analyze the review sentiment
Polarize the sentiment result
Collect result in CSV file
Load the result in pre-defined graph DB
Graph DB

4. Tweet data processing



Graph DB
Twitter stream arrived
Kafka creates Spark consumers
Spark Streaming receives live input data
Perform sentiment analysis for each message
Create batch file
Hourly
Load results to pre-defined graph db
company, sentiment result, tweet text, retweet amount etc.