Economy Monitor- W205 Final Project

Konniam, Evyatar, Daniel

12/15/2015

Problem

Prevailing economic conditions are crucial to organizations looking to make decisions that shape their futures, such as product launches, capital investment, hiring, and acquisitions. However, information which is important to the decision making process is often difficult to obtain. Many economic forecasts come once a quarter, and they are sometimes inaccurate. Decisions made on old, incomplete, or inaccurate information can lead to serious consequences for small and large businesses alike.

Traditionally, information on unemployment comes from sources such as unemployment claims, and business reports; this information is aggregated by government agencies, such as the Bureau of Labor Statistics (BLS); then it is disseminated in periodic reports to the public. Unemployment statistics from government agencies like the BLS are very important and useful; however, they have a number of issues. The reporting and aggregation process takes time, which means that even the most frequent reports can only be generated weekly. Also, these reports are rigid and follow a predefined schema. While this allows for easy reporting and comparison across time, it does not allow for inquiry outside of the schema.

Our project provides a way for businesses to gather near real-time insights on unemployment trends, through systematic parsing of social media posts. In particular, we gather “tweets”, or messages, from the social media application Twitter. The data is very unstructured, and it comes in with high velocity and volume. We process these tweets, searching for messages related to unemployment. When people post about someone losing their job, or a company doing layoffs, they are providing information that can be very valuable in the business setting. Our approach, makes it possible to update our statistics with this new information quickly so that users stay up-to-date. Since our data is raw, we are not bound by a rigid schema, which means users can make dynamic queries to find out information about business competitors, or local-level trends. Businesses can benefit from more frequent, and perhaps more locally detailed information, which can be streamed and visualized daily.

Data and Processing Dimensions

We used Twitter's Streaming API as a data source. Only a sample of all public tweets are available for consumption through this API. The velocity is on the order of 1.4 million tweets (6GB) per day, which should be easily handled by an appropriately configured AWS instance. Twitter's data is fairly unstructured. Although each tweet comes in a JSON format, not all fields are present, and any given field can have a variable number of items.

In terms of processing, because real-time processing is not necessary, we could process the tweets in batch. An hourly batch interval is likely good enough for any business question users want to answer with the data. There won't be advanced processing like machine learning at this time, though we would like to add this capability in the future.

Given the above dimensions, we decided to use MongoDB as our database. MongoDB is a document-based database, a very close match with our data set. The built-in analytic abilities are sufficient for our needs. MongoDB can be scaled out via horizontal sharding. While we did not do this in our project, we could easily implement it if the volume calls for it.

Architecture Overview

(Note: brief setup instructions are provided in setup.txt on Github)

AWS

- AWS allows for flexible scaling
- m3.xlarge instance with 15GB ram, 400GB EBS (1.2K/3K IOPS)
- Amazon Linux (Linux allows for precise command line control and easy scheduling)

Software

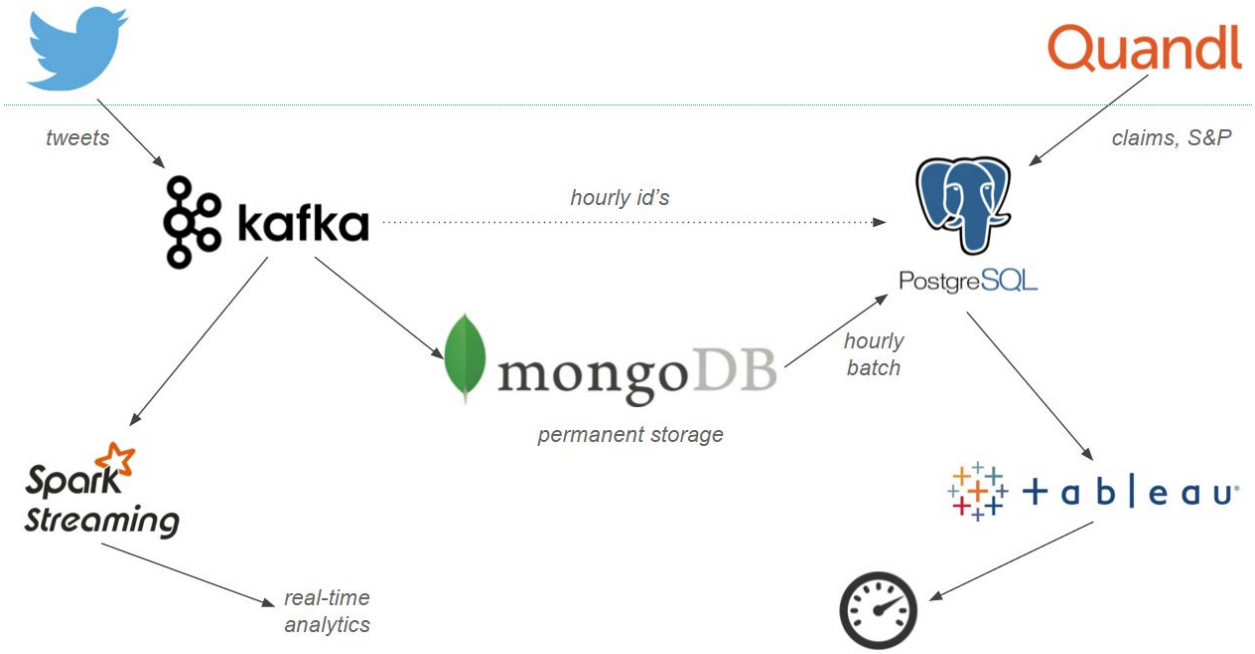
- Kafka, MongoDB, PostgreSQL, Spark

Python Packages

- tweepy, kafka-python, pymongo, psycopg2, Quandl

Data Collection

- 12/3 - 12/10 (10 million tweets, 42GB)



Data sources

- Twitter Streaming API and Quandl (economic and financial data)

Ingestion: Kafka

- Kafka decouples the production and consumption of messages, so multiple producers and multiple users could work together in harmony. In our case, if the database write to MongoDB slows, the Kafka layer allows the system to catch up.
- If we happen to add another data source in the future (that could be stored in MongoDB), we could simply create a Kafka topic and data will flow into our system.
- Kafka can be scaled out easily.
- In our current setup, we have 3 consumers connected to the Kafka producer. This made sure all downstream pipelines are seeing the same data.

Serving (PostgreSQL)

- We are using PostgreSQL as a serving layer for processed tweets. Because the processed data at the moment are simply keyword counts and frequencies, we could use a RDBMS to facilitate ad-hoc querying by SQL users.
- We are also using PostgreSQL to store stocks and economic data. Because we aren't storing stocks data by the millisecond, a SQL database is perfect for columnar data.

Spark Streaming

- Spark Streaming provides real-time batching capability. If our users want to see unemployment-related tweets in real-time (10-second window for instance), they could do so easily using Spark's streaming component. We simply have to connect the Kafka to Spark and we could be doing both hourly batches and real-time 10-second batches simultaneously.

Tableau

- We used Tableau to build a dashboard for the frequencies of keywords related to unemployment. This is a good way for business users to visualize trends.

Data Acquisition

Twitter provides an extensive API through which we can obtain tweets in real-time. We are however limited to 1800 tweets per every 15 minutes with the free option. Our approach is to ingest as much data as possible, given these limitations. The Python module tweepy provides streaming functionality, through which tweets are brought into our system. In order to protect against failover and data loss, our application transfers tweets to a Kafka queue, where they await next steps. Each of the above steps is taken care of by an ingestion script, twitter_ingest.py.

Next, the raw data from each tweet is converted to JSON form, and it is transferred to a MongoDB database; the key for each tweet is its previously assigned ID. These keys are also immediately stored in a postgres database, last_hour_tweets, because the most recent tweets will need to be counted and analyzed. By keeping track of the IDs for tweets from the last hour, we ensure that the tweets themselves can be retrieved rapidly from the MongoDB database. These steps are taken care of by a script called twitter_write.py.

In addition, we use the Python Quandl library to get stock quotes from S&P 500 companies. This data is numeric and structured, and we only pull it once a day. Therefore, it is suitable to be stored in Postgres.

Data Processing

Once the tweets are ingested, we are particularly interested in those that are related to unemployment. We use regular expression syntax to search for keyword patterns within each tweet. For example, the regular expression 'lost +(my|his|her|your|their)? *job' matches many different phrases, including "lost job", "lost their jobs", and "lost her job". The regular expression 'la(y|id)?[-]?off' matches the phrases "layoff", "laid off", and "lay-offs", among others. Therefore, each or our regular expression allows for more general searching.

We select the tweet IDs for the last hour from the last_hour_tweets Postgres database. We use these tweet IDs to obtain the tweets which were ingested during the last hour. As we find tweets that match our keyword patterns, we tally the results for each regular expression and store the counts (per hour) along with the hour in which they occurred in a Postgres table, keyword_tweets_cnt.

At the same time, we keep track of the total tweets occurring each hour in the Postgres table total_counts. Once the above steps are complete, we delete the tweet IDs from the last_hour_tweets table, to ensure that it doesn't grow too large. Note that we have not lost information, because the IDs are still contained in the full collection of tweets from the MongoDB database.

The steps above to process tweets are contained in the twitter_process.py script; this script can be run hourly in a cronjob, and it will process the tweets from the previous hour.

Data Analysis and Serving

The processed results reside in PostgreSQL tables. SQL users can easily query the database to analyze trends. For instance, they could join the keyword_tweets_cnt table and the total_tweets table (see Appendix for schema) to get frequencies of keywords. As we collect more data, we could join the claims and gspc tables as well, in order to predict claims (claims filed for employment benefits, a negative indicator for the economy) using machine learning. These SQL tables could be connected to Tableau as well. We have built a few dashboards that we think business users would be interested in.

If the keyword count tables and Tableau dashboards are not sufficient, the MongoDB database could be queried. Because the database contains all metadata of any tweet, one could easily process the saved data to extract additional insight.

Spark Streaming

The streaming component was added to the system for a few reasons. First, while unlikely, if people in our company want to see the unemployment keyword count data in real time, they could do so using our *twitter_spark.py* script (which relies on a Spark-Kafka package in order to run). The streaming component essentially has the same functionality as the batch component, but the time interval of aggregation is on the order of seconds instead of an hour. Second, it is very likely that there is a set of keywords, different from the batch process, that any company wants to track in real-time. For instance, TV show's producers might want to gauge how audience is feeling about the show. They may do so by tracking the variants of 'breaking bad awesome'. Our Spark streaming system could easily accommodate this use case. Users also has the option of persisting the results of that analysis to disk.

Next Steps

Our next steps would be to generate a series of models which can predict the above mentioned economic indicators. The features in the models would be the data collected thus far - focusing on word frequencies in tweets. The classification for the training would be the actual economic indicators. One concern with this approach would be the number of examples for the training. most economic indicators such as unemployment are monthly indicators, and so even if we take data for the last four years (assuming we could get the tweets for that period), we would only have about 50 data points for the training. On the other, the model itself would be pretty simple as the number of features which indicate word frequencies could potentially be fairly small. Other features that could be of value are the ratio of tweets sent during work hours (assuming that people tweet less during the work day, and an increase in the percentage of tweets sent during those hours could represent an increase in unemployment)

Appendix

Links

Github: https://github.com/KonniamChan/economy-monitor

S3: s3://konniamchan-w205/ (permission set to any AWS authenticated user)

Contents of S3

GSPC.csv (S&P500 data)

claims.csv (initial jobless claims data)

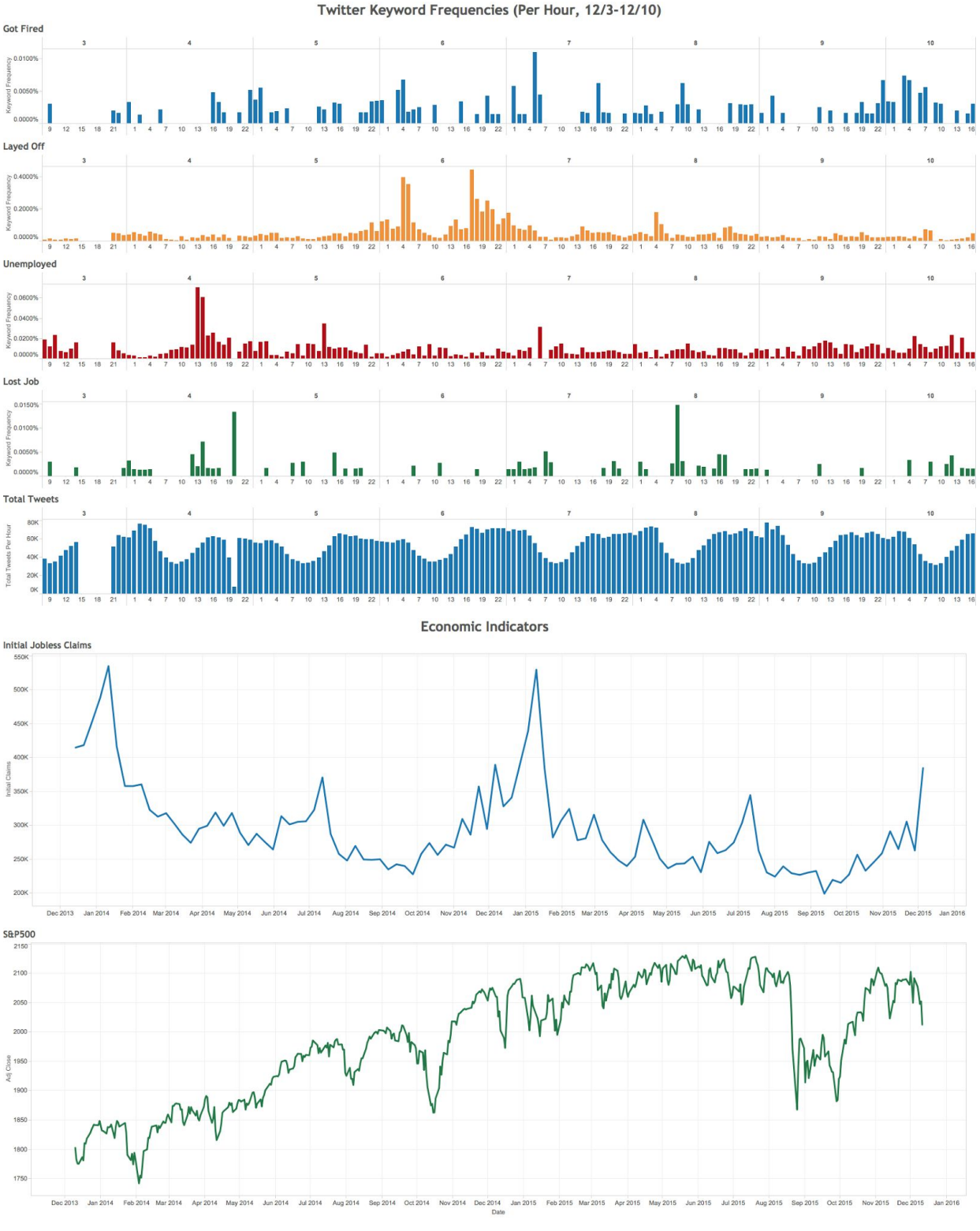
keyword_tweets_cnt.csv, total_tweets.csv (tables from PostgreSQL)

tweets_00.json.gz to tweets_09.json.gz (10M tweets split up into 10 compressed files)

Schema for PostgreSQL

```
-- list of tweet _ids for the last hour
CREATE TABLE last_hour_tweets (
    timestamp TIMESTAMP NOT NULL,
    tweet_id VARCHAR(50) NOT NULL PRIMARY KEY
);

-- hourly counts with keywords over last hour
CREATE TABLE keyword_tweets_cnt (
    timestamp TIMESTAMP NOT NULL,
    kword_search VARCHAR(100) NOT NULL,
    count INT NOT NULL,
    PRIMARY KEY (timestamp, kword_search)
);

-- hourly counts for total tweets over last hour
CREATE TABLE total_tweets (
    timestamp TIMESTAMP NOT NULL PRIMARY KEY,
    count INT NOT NULL
);

-- FRED initial claims data
CREATE TABLE claims (
     date DATE NOT NULL PRIMARY KEY,
     initial_claims REAL NOT NULL
);

-- S&P500 index data
CREATE TABLE GSPC (
     date DATE NOT NULL PRIMARY KEY,
     index_value DOUBLE PRECISION NOT NULL
);
```

Dashboards


Twitter Keyword Frequencies (Per Hour, 12/3-12/10)
Got Fired
Layed Off
Unemployed
Lost Job
Total Tweets
Keyword Frequency
Total Tweets Per Hour
Economic Indicators
Initial Jobless Claims
Initial Claims
S&P500
Adj Close
Date