Large Scale Sentiment Analysis of Tweets

James Tan Juan Whei 40161156
Vithushon Maheswaran 27052715
Saeedeh Moghimi 40184202
AmirAli Sabaghpourfard 40185709

Concordia University

August 13, 2022

Abstract

The aim of this paper is to analyze sentiments of tweets in a specific duration of time. We have used library called snnsrcape to scrape approximately one million tweets with size of 10GB and Apache spark and BigQuery to store and analyze the data. The library we have used to classify the tweets is called ClassifierDLModels. The results shows the percentage of sentiment in tweets during a certain time and also as the was a shooting in this period, charts shows how much sentiment of fear has been increased from one day to another. Finally, We conclude the results and show the charts and results.

1 Introduction

This project aims to apply distributed system concepts with real applications, such as NoSQL systems for big data, and demonstrate the power of distributed systems in extracting and analyzing the general sentiment expressed in a large data set of tweets, and detecting emotions and cyberbullying using pretrained Spark NLP DL classifiers. Emotions identified include: Joy, Surprise, Fear, Sadness while detecting Racism, Sexism or Neutral tweets.

2 Tools and Technologies

Given the large dataset, approximately 10 GB of scrapped tweets, we chose google cloud storage. Google Cloud Storage is a RESTful online file storage web service for storing and accessing data on a Google Cloud Platform infrastructure. The service combines the performance and scalability of the cloud with advanced security and sharing capabilities, all at no cost to a user using less than 15 GB of cloud space for free.

Furthermore, Big Query was chosen to store the tweets as tabular data in datasets and tables. BigQuery is a data warehouse and a SQL Engine that enabled ease of data import due to it's compatibility with GCS. It's also an affordable big data warehouse, where the first Terabyte of data processed each month is free.

In order to process our data and draw insight from processed tweets, we used Apache Spark, an open-source unified analytics engine for large-scale data processing. To run Apache Spark, we used Dataproc, given the fast, easy-to-use, and fully managed cloud service for running Apache Spark clusters in a simpler, more cost-efficient way. It helps you create clusters quickly, manage them easily, and save money by turning clusters off when you don't need them.

2.1 GCS

GCS, or Google Cloud Storage[4], is an object storage service on the Google Cloud. GCS stores a variety of data (objects) in containers called buckets. GCS offers different types of classes like Standard, Nearline, Coldline, and Archive, which make a difference in price and availability[5].

For this project, we used GCS to store CSV files before loading them into BigQuery. BigQuery can load documents from local and GCS, but since we wanted one table in BigQuery and big size of data files, it wasn't a good idea to store documents locally and make BigQuery load data from local. To decrease time consumption, we uploaded all the CSV files to a Standard class GCS, and then BigQuary loaded data from there. In order to upload files to GCS, we used this command:

gsutil cp *.csv gs://tweets_bucket_comp6231/

2.2 BigQuery

Google BigQuery is a cloud-native, enterprise-grade data warehouse. BigQuery was first made available as a service in 2010 and became generally available in November 2011. Since its debut, BigQuery has developed into a more affordable, fully-managed data warehouse that can process petabyte-scale datasets with lightning-fast interactive and ad-hoc queries. BigQuery is now more functional thanks to its integration with a number of Google Cloud Platform (GCP) services and third-party tools.

BigQuery, or more specifically data warehouse as a service, is serverless. No servers or database software needs to be installed or managed. The BigQuery service maintains the infrastructure, including scalability and high-availability, as well as the underlying software. The pricing structure is fairly straightforward: Every TB of processed data is 5 Dollars. BigQuery reveals a simple client interface which enables users to run interactive queries.

BigQuery is built on Dremel[1], which has been operating internally at Google since 2006. Dremel is Google's ad hoc interactive query system for analyzing read-only nested data. The first Dremel paper was published in 2010. At the time of publication, Google was running multiple instances of Dremel ranging from tens to thousands of nodes.

BigQuery and Dremel have the same underlying architecture. By integrating Dremel columnar storage and tree architecture, BigQuery offers unprecedented performance. But BigQuery is much better than Dremel. Dremel is just an execution engine for BigQuery. In fact, the BigQuery service uses Google's innovative technologies such as Borg[3], Colossus, Capacitor, and Jupiter.

A BigQuery client (usually the BigQuery web UI or bg command line tool or REST API) interacts with the Dremel engine through the client interface. Borg, Google's large cluster management system, allocates computing capacity to Dremel jobs. The Dremel job reads data from Google's Colossus file system over the Jupiter network, performs various SQL operations, and returns the results to the client. Dremel implements a multi-level server tree to execute queries.

The most expensive part of any big data analytics platform is almost always disk I/O. BigQuery stores data in a columnar format called capacitor[2] . As you can imagine, each field of BigQuery table i.e. column is stored in a separate Capacitor file which enables BigQuery to achieve very high compression ratio and scan throughput.

2.3 Dataproc and Spark

Dataproc is a managed framework that runs on the Google Cloud Platform and ties together several popular tools for processing data. We used it to run Apache Spark clusters. It's easy-to-use and relatively fast, as you can spin up a cluster in about 90 seconds. It's cheaper than building your own cluster because you can spin up a Dataproc cluster when you need to run a job and shut it down afterward, so you only pay when jobs are running. Dataproc is priced at only 1 cent per virtual CPU in your cluster per hour. Conveniently, it's integrated with other Google Cloud services, including Cloud Storage, BigQuery, and Cloud Bigtable, making it easy to transfer our data. Due to performance considerations as well as the high reliability of storage attached to Dataproc clusters, the default replication factor is set at 2.

Compute Engine Virtual Machine instances (VMs) in a Dataproc cluster, consisting of master and worker virtual machines, require full internal IP networking cross connectivity. The default Virtual Private Cloud network provides this connectivity, and offers native Internal TCP/UDP Load Balancing and proxy systems for Internal HTTP and HTTPS Load Balancing.

Dataproc clusters can be created using the Google Cloud Platform console, or alternatively to create a Dataproc cluster on the command line, run the Cloud SDK gcloud dataproc clusters create command locally in a terminal window or in Cloud Shell. Refer to section 7 - Code to see our code for cluster creation.

Basic requirements for cluster creation include specifying a cluster name, region (global or a specific region for the cluster), and connectivity consisting of the number of master and worker nodes.

gcloud dataproc clusters create cluster-name –region=region

The above command creates a cluster with default Dataproc service settings

for your master and worker virtual machine instances, disk sizes and types, network type, region and zone where your cluster is deployed, and other cluster settings.

In order to set attributes specific to the required resource, you can specify the specific values against the flag you want to modify.

The following flags were used to specify unique attributes for the resources of our VMs:

Type of machine to use for worker nodes.

–worker-machine-type n1-standard-8

The size of the boot disk, in our case a 128 GB disk.

–master-boot-disk-size=128GB –worker-boot-disk-size=128GB

We used the default number of master nodes (1 node), however, to specify otherwise when High Availability is critical, the number of master nodes in the cluster can be changed to 3.

–num-masters=1

The number of worker nodes in the cluster, we only used 2 worker nodes.

–num-workers=2

To submit a job to a Dataproc cluster, run the gcloud CLI gcloud dataproc jobs submit command locally in a terminal window or in Cloud Shell.

3 Dataset

We relied heavily on the snscrape [10] library to scrape tweets. We filtered for tweets that were in the English language only. Approximately 1000000 tweets were scraped for each day between 2022-05-01 and 2022-07-12. The tweets were then saved in Google Cloud Storage.

4 Processing of Data

As mentioned in Section 2.3, we define the processing of the data as a Spark job. The steps involved in the job are illustrated in Figure 1 and will be further elaborated on in the subsequent sections. Our processing pipeline relies heavily on the Spark NLP library[9].

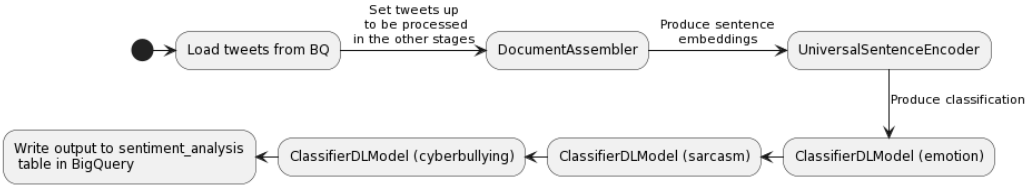

Figure 1: The processing pipeline

4.1 Data Ingestion

In order to use data stored in BigQuery as an input to our Spark job, we used the Spark BigQuery connector[6]. The Spark script reads from a table that contains all the tweets that were procured as described in Section 3.

Each run of the Spark job would typically be executed on 4-5 days' worth of tweets as we discovered that the Spark jobs had a tendency of failing when working with larger amounts of data. This was true even when the CPU and memory utilization of the worker nodes were relatively healthy and thus should be further investigated.

4.2 Document Assembler

The first step of the pipeline is the `DocumentAssembler`[8]. This prepares the data into a format that is processable by Spark NLP and is essentially the entry point for every Spark NLP pipeline.

4.3 Generation of Sentence Embeddings

We generate sentence embeddings by leveraging a Universal Sentence Encoder[11] made available by Tensorflow. The output of this stage is a 512-dimensional vector that semantically captures the meaning of each tweet. This is the basis upon which the downstream classification algorithms build on.

4.4 Sentiment Classification

To actually use the embeddings described in the previous section, we utilise ClassifierDLModels[7] to classify the tweets. Each ClassifierDLModels essentially assigns a label to each tweet. To identify the emotion, presence of cyberbullying and presence of racism in each tweet, we use the `classifierdl_use_emotion`, `classifierdl_use_cyberbullying` and `classifierdl_use_sarcasm` pretrained models respectively.

The emotion classifier produces the values `sadness`, `joy`, `love`, `anger`, `fear` and `surprise`. The cyberbullying classifier produces the values `neutral`, `racism` and `sexism`. The sarcasm classifier produces the values `sarcasm` and `normal`.

4.5 Storing of Output

The output is then stored in a separate table in BigQuery. Note that the BigQuery Spark connector is once again used here, thus allowing the output of a Spark job to be appended directly to a BigQuery table.

5 Results

To visualize the general sentiment expressed in tweets, detected emotions and cyberbullying, we used Jupyter notebook to display and provide functionality to query the data interactively.

Emotions identified include: Joy, Surprise, Fear, Sadness while detecting

Racism, Sexism or Neutral tweets.

Below you see the sample output for a query on racist tweets between the 1st and 4th of May, 2022. Note, the data output has been cropped to 7 tweets for convenience.

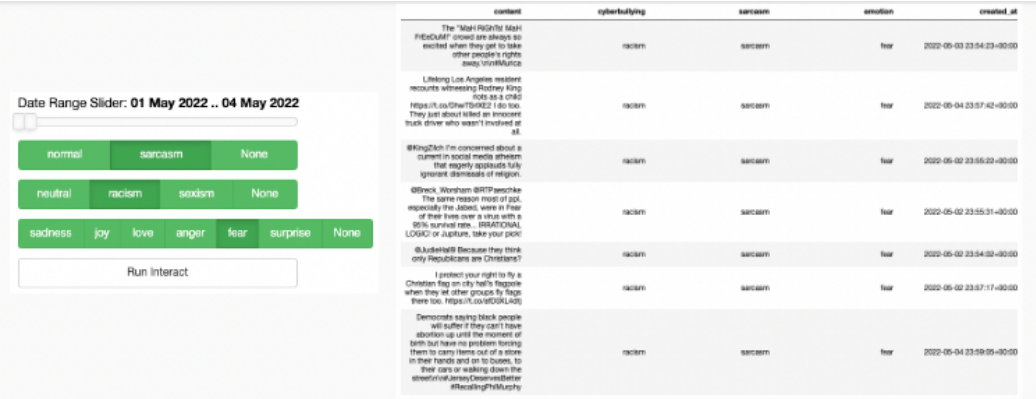

Figure 2: Sample Query: Racist Tweets

Below is another visualization provided using Jupyter notebook, that enables us to visualize the emotions of tweets over time. Interestingly, you can see the spike of fear on May 24th, 2022.

6 Future Work

An idea that we originally had was to run an unsupervised clustering algorithm on the sentence embeddings produced in Section 4.3. The clustering algorithm that we had in mind was DBScan[13], with the objective of clustering tweets with similar topics. Unfortunately, we were not able to find a satisfactorily efficient implementation of the algorithm to employ with Spark. Due to the size of our dataset and the high number of dimensions of the embeddings, an efficient implementation was crucial to the success of this idea. Given that we had limited resources, we were forced to abandon this idea, however, it should be revisited in the future by implementing our own version of the DBScan algorithm.

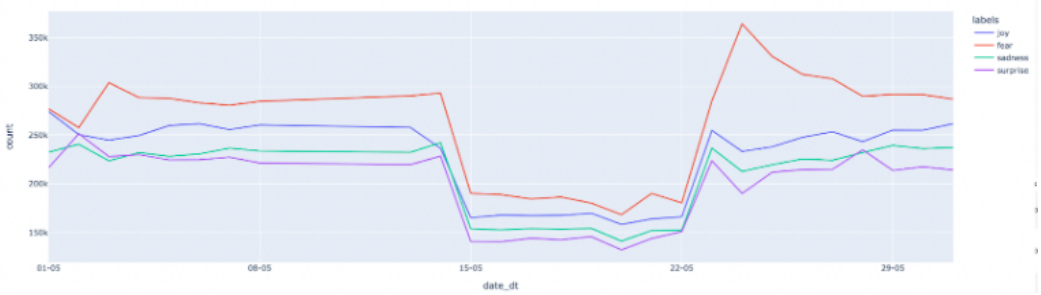

Figure 3: Sample Query: General Sentiment Over Time

By using our interactive tweet-emotion visualizer, we can observe the emotion of the general users on twitter, and here we unfortunately notice that a shooting had taken place on the 24th of May, 2022.

content	cyberbullying	sarcasm	emotion	created_at
@richardmarx The ones who say that are the ones who value their guns over people lives. Sick twisted country we live in	racism	sarcasm	fear	2022-05-26 23:55:59+00:00
All citizens of America want to see those cops dangling!\n\nMaybe the most unification I've seen on an issue in a long time. That part at least	racism	sarcasm	fear	2022-05-26 23:54:31+00:00
We are FAILING the children of this country. It shouldn't have to be this way. I have given up expecting anything to change. Nothing will change except the rally cry of gun fanatics bellowing "but not me" louder and louder and louder.\n\n#Uvalde	racism	sarcasm	fear	2022-05-24 23:56:16+00:00
@CNN Evil people do evil shit.	racism	sarcasm	fear	2022-05-14 23:54:51+00:00
@Ellenleigh3 @Jay_McGill94 so cops are ok with white people shooting them?	racism	sarcasm	fear	2022-05-24 23:58:07+00:00

Figure 4: Sample Query: Fear

7 Code

The code to reproduce our project may be found in our Github repository [12].

8 Team-Work

Given the limited time, the project scope was distributed amongst each team member, and logically split into three distinct areas as follows:

1. Data storage
2. Data processing
3. Data visualization

Saeedeh was responsible for storing our raw data into BigQuery Table. James and Vithu were responsible for processing the data. AmirAli was responsible for the visualization of the data.

9 Conclusion

We explored various distributed applications that stored, processes, and analyzed data, while considering multiple factors such as performance, cost, ease-of-use, availability, etc. This project has demonstrated the convenience of such applications based on Google Cloud architecture. Although, our data processing was not as compute intensive to warrant the use of the clusters chosen, the project has demonstrated the capabilities of distributed systems by extracting, analyzing, and storing the general sentiments expressed by the large data set of tweets.

References

[1] Google. Dremel: Interactive analysis of web-scale datasets, 2010. `https://research.google/pubs/pub36632/`,.

[2] Google. Storing and querying tree-structured records in dremel, 2014. `https://research.google/pubs/pub43119/`,.

[3] Google. Large-scale cluster management at google with borg, 2015. `https://research.google/pubs/pub43438/`,.

[4] Google. Gcs, 2022. `https://cloud.google.com/storage`,.

[5] Google. Gcs classes, 2022. https://cloud.google.com/storage/docs/storage-classes,.

[6] Google. Use the bigquery connector with spark, 2022. https://cloud.google.com/dataproc/docs/tutorials/bigquery-connector-spark-example,.

[7] John Snow Labs Inc. Classifierdlmodel, 2022. https://nlp.johnsnowlabs.com/api/com/johnsnowlabs/nlp/annotators/classifier/dl/ClassifierDLModel.html,.

[8] John Snow Labs Inc. Documentassembler, 2022. https://nlp.johnsnowlabs.com/api/com/johnsnowlabs/nlp/DocumentAssembler,.

[9] John Snow Labs Inc. Spark nlp: State of the art natural language processing, 2022. https://nlp.johnsnowlabs.com,.

[10] SNScrape. Scraper for social networking services (sns), 2018. https://github.com/JustAnotherArchivist/snscrape,.

[11] Tensorflow. Universal sentence encoder, 2022. https://www.tensorflow.org/hub/tutorials/semantic_similarity_with_tf_hub_universal_encoder?hl=en,.

[12] COMP6231, Summer 2022, Team 8. Github repository of the project, 2022. https://github.com/jamestjw/tweet_analysis_comp6231,.

[13] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.