CS 631: Final Project

Real-time sentiment analysis of tweets

Github: https://github.com/chrisbombino/cs631-project

Shahzeb Naveed
Management Sciences
s6naveed@uwaterloo.ca

Yuqing Zhang
Data Science and AI
y3593zha@uwaterloo.ca

Christopher Bombino
Management Sciences
cjbombin@uwaterloo.ca

ABSTRACT

Twitter is a popular social networking website where users interact with each other using short messages referred to as tweets. Every second, over 350,000 tweets are generated [1] from around the world in which users share their views about various trending and non-trending topics, communicate with friends, brands, or different services, or share content about their daily lives. Thus, the information available on the platform can prove to be very useful in understanding the users' opinions and sentiments about general trends across various socio-demographic factors. Particularly, it may prove to have invaluable information that companies across industries can leverage to better understand their customer's satisfaction with their products and/or gauge it against the popularity of counterpart products of their rival companies. In this project, a real-time system for streaming, sentiment analysis, and visualization of live tweets is presented that can be very handy for the purpose of market research. The live twitter data will be streamed using Kafka, analyzed using NLP/Deep Learning methods, stored in Elasticsearch, and then analyzed in a dynamic Kibana dashboard. The system was successfully implemented and was put into production on AWS.

CCS CONCEPTS

• Computing methodologies • Artificial Intelligence • Machine Learning • Distributed Computing Methodologies • Applied Computing • Document management and text processing • Information Systems • Information Retrieval

KEYWORDS

Kafka, Elasticsearch, Kibana, Twitter, Big data, AWS, Machine Learning, Deep Learning, Natural Language Processing

1 Objective

The goal of this project is to analyze the sentiment towards a number of companies using real-time tweets. The data will be streamed using Kafka, analyzed using NLP/Deep Learning methods, stored in Elasticsearch, and then analyzed in a Kibana dashboard.

The learning outcomes can be summarized below:

- General Kafka architecture - topics, producers, consumers
- Applying natural language processing and deep learning to perform sentiment analysis of text data
- Indexing and retrieving data in Elasticsearch
- Developing dashboards with Kibana
- Running everything on AWS EC2
- Automating the process with Terraform

2 Methodology

The system (as shown in figure 1) consists of multiple components that play their part in sourcing, logistics, analysis, storage, visualization, hosting, and management, each described in detail below.

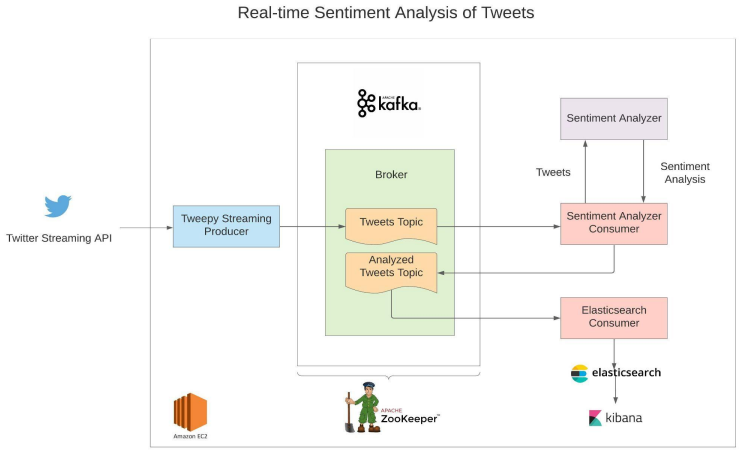

Figure 1: System Architecture

3 Implementation

3.1 Apache Kafka and Zookeeper

3.1.1 Introduction to Kafka

Apache Kafka is an open-source distributed, highly scalable, fault-tolerant event streaming platform that has made its mark in the realm of streaming analytics and data integration [2]. A **Kafka cluster** comprises a number of servers, some of whom form the storage layer and are referred to as **Kafka brokers** while others are responsible for transporting data into and out of the cluster.

A **Kafka topic** is a logical division of data that helps us put similar data in similar locations in the cluster. The data in the topics is further partitioned and stored in a distributed manner in the Kafka brokers. The data is published by various sources referred to as the **Kafka producers** onto relevant topics, which the **Kafka consumers** can subscribe to. This essentially decouples the producers and consumers making it act like a message broker or a buffer, and this is how it avoids data loss in the case any one of them crashes, the very reason this tool was included in the system architecture. Furthermore, its distributed nature helps store streaming data in a durable manner and the whole architecture can be leveraged to process data on the go as well using add-on modules such as Kafka Streams API, an alternative to the popular Spark Streaming API.

3.1.2 The Zookeeper

The Kafka cluster is itself managed by **Apache Zookeeper**, another top-level Apache project. Some of its tasks include managing synchronization, keeping track of the status of Kafka servers and configuration of topics including maintaining a list of existing topics, number of partitions and location of replicas, etc. [3]

For the purpose of the project, the system comprises a single broker and two different topics, storing the raw tweets and analyzed tweets respectively.

3.2 Tweepy Streaming Producer

Twitter Streaming API provides an interface to retrieve live tweets from Twitter relevant to a predefined set of keywords. The following keywords were chosen in order to be able to draw comparisons of sentiments of different products of Apple and Samsung:

- iPhone
- iPad
- Apple Pay
- Macbook
- Apple Watch
- Airpods
- Samsung Galaxy S20
- Galaxy Tab
- Samsung Pay
- Samsung Laptop
- Samsung Galaxy Watch
- Samsung Galaxy Buds

In this module, the Tweepy Python package that provides a StreamListener class was used to interact with the API. In addition to the raw text of the streaming tweets, the API provides metadata as well. The following data were extracted for every streamed tweet:

- text
- created_at
- id
- hashtags
- user_id
- user_location
- user_description
- user_followers_count
- user_friends_count
- user_listed_count
- user_favourites_count
- retweet_count
- favorite_count

The module acts as a Kafka producer and publishes data to the Tweets Topic in the Kafka cluster.

3.3 Sentiment Analyzer Consumer

The Sentiment Analyzer Consumer module subscribes to the Tweets Topic and is responsible for performing sentiment analysis with the help of the Sentiment Analyzer module discussed in detail below. This module is also responsible for performing pre-processing of the text data and identifying the associated company and product for every processed tweet.

The results of this consumer are published back to a separate Kafka topic, the Analyzed Tweets Topic. The reason for this is the same, to decouple applications to avoid data loss in the event of an application crashing or a networking issue.

3.4 Sentiment Analyzer

There are mainly two steps in building the sentiment analyzer. The first is converting tweet text to an integer token sequence. The second is using the neural network model, which we trained in other static dataset and already

saved, to predict the sentiment into three categories; negative, neutral, and positive.

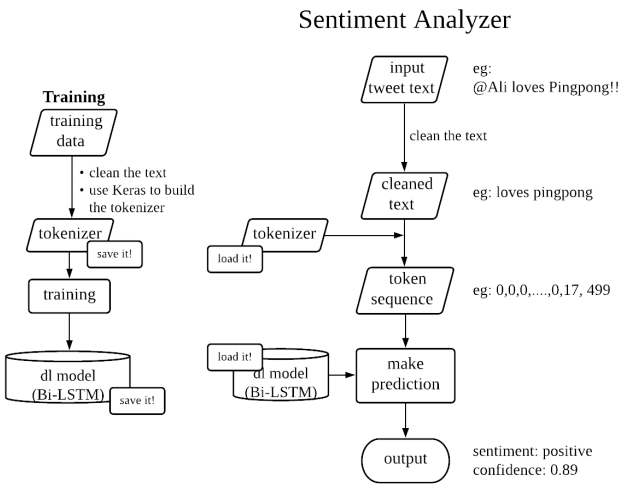

Figure 2: System Of Sentiment Analyzer

3.4.1 Text Preprocessing - cleaning

Among all the steps, the first important one is text preprocessing. Before that, we have already made sure only tweets in English languages are gathered. However, the text data still contains non-English characters, emojis, punctuations, and some other stuff that may interfere with the predictions. Text preprocessing includes several steps:

- cleaning data and only keep English letters: removing username, URL, numbers, special symbols, and other non-English characters
- converting text into lower case
- removing stopwords

The same text preprocessing method was used in both training data and input tweet text to be predicted in the project.

3.4.2 Text Preprocessing - vectorization

The **text tokenization** utility class in Keras is a good tool for text vectorization. We use the large training dataset to build the tokenizer. It will form a token-index dictionary where the key is each token and the value is the corresponding index. We then save the tokenizer and load it into the program when we do the sentiment prediction.

The tokenizer can turn each text into a sequence of integers, where the integer represents the index of the token in the whole dictionary. Eventually, we pad the sequence into a 2D array of shape (number of examples, max length). Since each tweet can contain at most 140 characters, it will not cover too many words after removing stop words and doing the text cleaning. Therefore we define the max length of sequence 32. Then sequences that are shorter than 32 are padded with 0 until they are 32 long. And for the tweet longer than expected, we will truncate the number list so that they fit the desired length.

Here illustrates the above two processes using an example: for one tweet text "RT @userA this is an amazing website!! http://xxx.com". After cleaning the data, it will turn into a short text sequence as "amazing website". Then we use the tokenizer to convert it to a 32-length array, where the first 30 numbers will be 0 and the final two elements will be the index of 'amazing' and 'website'.

3.4.3 Dataset

There are various datasets related to sentiment analysis available online. However, some of them are too small and only focus on certain topics (like tweets about airline service). If we train on these datasets, we can obtain classification models with high accuracy, but overfitting and bad generalization can be easily observed. They perform badly on other datasets, and not to mention real-time tweet data. Additionally, some of the datasets only have data in two polarities: negative and positive. However, in reality, countless tweets do not have strong semantics and we cannot simply divide them into positive or negative.

The datasets found by far all have certain flaws so that it is hard to only pick one of the datasets. As the objective of the project is to analyze the sentiment of all the tweets we may encounter in real life, we decided to merge some of the high-quality datasets and made a selection so that the class labels are balanced.

The datasets we used included:

- Kaggle: Twitter US Airline Sentiment [10]
- Kaggle: tweet sentiment extraction [11]
- Kaggle: preprocessed Twitter tweets [12]
- Apple Twitter sentiment [13]
- Sentiment 140 [14]

In total, a dataset of 60115 tweets is gathered. We applied the preprocessing function in all the text and kept the clean version of the text. Then we use the dataset to generate the tokenizer, i.e. the word-index dictionary. The tokenizer will be used in each prediction to ensure the same token in all the input tweets can be converted into the same index integer. We only load the tokenizer once before all the analysis.

The dataset is balanced, with 34% of them are labeled 'positive', 34% 'negative' and 32% 'neutral'. The dataset we created is large enough to be appropriate to train. Admittedly, it still has some limitations. Since it was created by combining several datasets and each dataset has its own text style, it is hard to generate a perfect model.

3.4.4 Model

In this project, a **bidirectional long short-term memory network (Bi-LSTM)** is proved to be the best. The bidirectional layer is the key determinant of the high performance, which maximizes the sequence sensitivity of RNNs. As the name suggests, the layer consists of two LSTMs that process the input sequence in a different direction to the final combined representation. In other words, the two layers interpret the sequence in chronological order and reverse order respectively. In this way, this model is more capable of capturing complex patterns than a single **RNN** layer. [5]

The whole model architecture is not complicated. And it will save the number of parameters largely than one-way LSTM because of the introduction of the bidirectional layer.

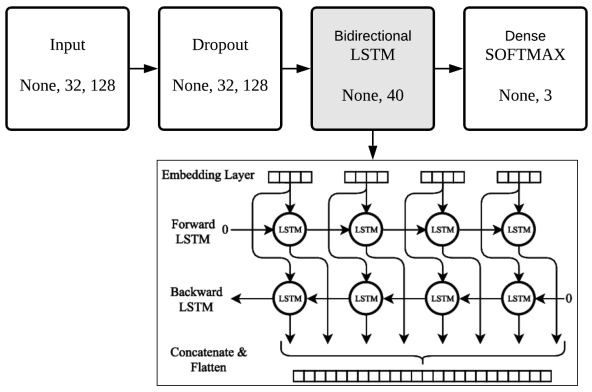

Figure 3: Model Architecture -BiLSTM (the below part of figure from [15])

3.4.5 Implementation

For the generation of deep learning models, Keras is used to train the deep learning model. Other than the traditional method, Pyspark is also tried during the implementation.

In the **PySpark** method, a Keras model can be constructed as before. The key differences are to load the data as an RDD and fit the data into a PySpark pipeline containing the Keras model. A necessary package is Elephas, an extension of Keras. It allows to train and run deep learning models in a distributed way at scale with Spark.

The implementation using Pyspark and Elephas was successful and a model can be trained as the local machine. However, the drawback of this Spark method is that we can only save the fitted pipeline, not the model. It is not convenient for further implementation of this project. Also, the prediction process is slower than the general model due to the long internal process of spark, which is a hurdle to the real-time prediction. So for now this Spark method is not used. But it is undeniable that this is an innovative and useful method. It is worth exploring more using Spark in the future.

3.5 Elasticsearch Consumer

This module subscribes to the Analyzed Tweets Topic and is tasked with responsively dumping the processed tweets into Elasticsearch. Additionally, the module is responsible for creating the **Elasticsearch index** and defining a mapping for the created_at field of the index, so Elasticsearch is aware this field is a date and not simply a string.

3.6 Elasticsearch

Elasticsearch is a search engine based on the Apache Lucene project that provides distributed, full-text search capabilities. The highest level of abstraction in Elasticsearch is a cluster; this can be conceptualized as a group of servers that all work together to serve a dataset to a client. Clusters contain indices, which then contain documents. A document is the smallest unit in Elasticsearch and comprises multiple fields. Compared to a relational database, a document is similar to a row and an index is similar to a table.

Elasticsearch has gained popularity over the years and has a thoughtfully designed REST API, as well as libraries for the most popular programming languages. Another primary reason for Elasticsearch's popularity is how easily it can integrate with Kibana (discussed below), a data visualization dashboard built specifically for Elasticsearch.

3.7 Kibana

Kibana provides a web user interface to manage and visualize Elasticsearch. An index pattern corresponding to the index where the processed tweets reside in the Elasticsearch was specified and then various visualizations

were developed and integrated on a dashboard. The visualizations display:

- The total number of tweets analyzed
- The user sentiments of a company over time
- The breakdown of general sentiments
- The breakdown of sentiments about products by percentage of tweets
- The table consisting of individual tweets along with the predicted sentiments, the confidence levels of the predicted sentiments on a scale of 0 to 1, and the names of the associated products and companies.

In addition to the visualizations, the following controls are provided for the user to filter the results:

1. Company Filter
2. Product Filter
3. Hashtag Filter

A snapshot of the Kibana dashboard is shown in the fig. 4 and can be accessed live at tinyurl.com/4uyxenaw. Please log in using the following:

- username: elastic
- password: KZcHE52wSJaEs5i8fk3A

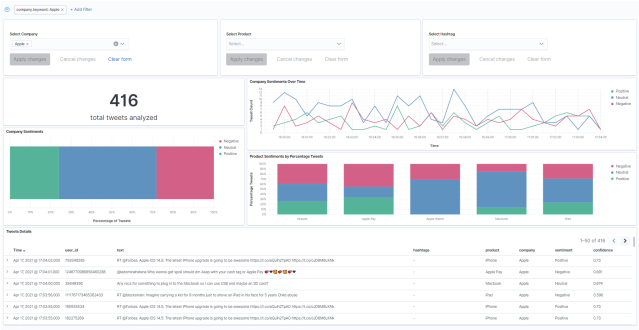

Figure 4: Kibana Dashboard Screenshot

3.8 Productionizing The Application

Productionizing the application requires setting up a server that can be accessed over the internet. **Amazon Web Services (AWS)** was the chosen cloud provider to host the application due to the ease of setting up a server (known as an EC2 instance). The process of configuring the server was similar to setting up a local development environment: install the necessary software, start the producers and consumers, then view the Kibana dashboard. The live server has some extra complexities because the aforementioned processes will run indefinitely, as opposed to eventually stopping the services on your laptop. Additionally, AWS's custom AMI feature was used to create an OS image that already had Kafka, Zookeeper, Elasticsearch, and Kibana installed with the required configurations for each piece of software. That way on each new deployment, the only work required was to clone the source code from GitHub, install the Python dependencies, and run the producer/consumers. This allowed for standardizing the configuration of the system so it could be repeated in a consistent manner. To manage the deployment itself, a tool called Terraform was utilized so that the final infrastructure and configuration setup could also be stored and versioned as code, instead of having an individual be in charge of performing the final configuration (this follows the Infrastructure as Code methodology).

4 Results

The system was successfully implemented and was put into production on AWS. The Sentiment Analyzer's prediction results are discussed below in detail.

The best model so far has been the Bidirectional LSTM. It is slightly better than the simple LSTM model and largely outperforms the traditional CNN model. From the line chart (Fig.5), Bi-LSTM (the blue line) has the highest validation accuracy (71.12%) in the 7th epoch than any other models. Therefore we save this model for further use.

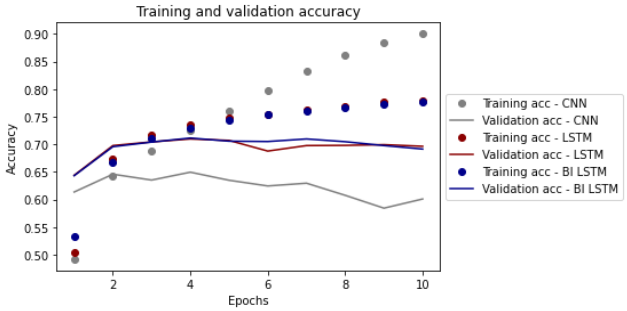

Figure 5: Training and Validation Accuracy

From the figure of the confusion matrix (Figure 6) and the model performance metrics table (Table 1), we can deduce that the model performs well in classifying positives and negatives, but a bit poorly in detecting neutral from others. This is largely due to the internal shortcomings of training datasets. In figures, 79% of negative examples and 75% of positive examples are predicted correctly, and. only 9% of positive ratings were classified as negative and 7% of negative ratings were classified as positive. And in this project, neutrality is of less importance than the other two categories.

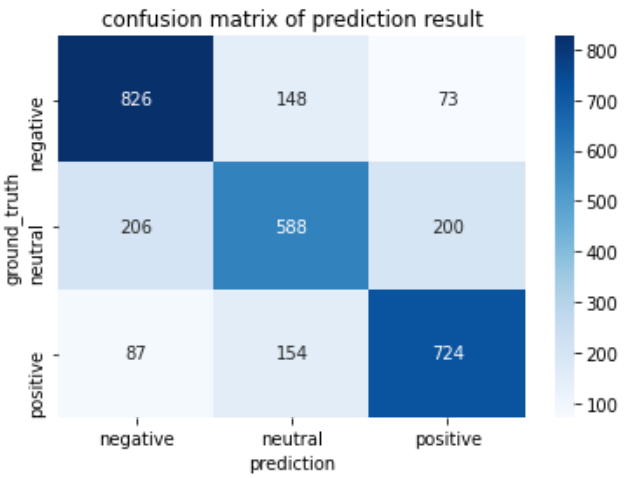

Figure 6: Confusion Matrix Of The Prediction Result

Table 1: Model Performance Metrics on test set

	Precision	Recall	F1 Score	Support
0: negative	0.79	0.74	0.76	1119
1: neutral	0.59	0.66	0.62	890
2: positive	0.75	0.73	0.74	997
Accuracy	0.71	0.71	0.71	3006
Macro Average	0.71	0.71	0.71	3006
Weighted Average	0.71	0.72	0.71	3006

5 Learnings/Shortcomings

In summary, a system was established successfully in this project, where real-time Twitter data can be streamed using Kafka, processed and analyzed using deep learning methods, stored in Elasticsearch, and then visualized in a Kibana dashboard. One of the biggest learnings is to build a big data system with potential commercial value.

One of the issues faced during testing of the system was that the producers and consumers sometimes end up processing the same tweet multiple times, which results in duplication. Although this duplication was handled separately in the Elasticsearch Consumer application, the Kafka framework can be studied further in detail to understand how the offsets work in case of failures and how **exactly-once semantics** can be implemented to avoid duplication. This issue may arise because of a broker failure or producer-to-broker communication failure when the broker is able to meet the request of writing data into a topic but is not able to acknowledge (referred to as an ack) to the producer which may re-try writing the same tweet as a result. [4]

During the implementation, there was some delay between producer and consumer because the tokenizer was being fitted every time the Consumer Sentiment Analyzer script was run. Because the tokenizer is fitted on a static training dataset, it makes more sense to have it trained only once and then pickle it for future use. This will minimize the time taken by the Sentiment Analyzer to make predictions.

ACKNOWLEDGMENTS

We would like to thank Professor Ali Abedi and the TAs of the course CS 631 for laying a strong foundation of various concepts of data intensive distributed computing for us and helping us to familiarize ourselves with various big data frameworks throughout the course.

REFERENCES

[1] https://www.internetlivestats.com/twitter-statistics/
[2] https://kafka.apache.org/
[3] https://www.cloudkarafka.com/blog/cloudkarafka-what-is-zookeeper.html
[4] https://www.confluent.io/blog/
[5] An easy tutorial about Sentiment Analysis with Deep Learning and Keras https://towardsdatascience.com/an-easy-tutorial-about-sentiment-analysis-with-deep-learning-and-keras-2bf52b9cba91
[6] Pyspark Elephas, an extension of Keras https://github.com/maxpumperla/elephas
[7] Distributed Deep Learning Pipelines with PySpark and Keras https://towardsdatascience.com/distributed-deep-learning-pipelines-with-pyspark-and-keras-a3a1c22b9239
[8] Sentiment Analysis on US Twitter Airlines dataset: a deep learning approach https://devklaus.wordpress.com/2018/03/11/sentiment-analysis-on-us-twitter-airlines-dataset-a-deep-learning-approach/
[9] A twitter sentiment analysis pipeline with neural network, kafka, elastic search and kibana https://devklaus.wordpress.com/2018/05/03/a-twitter-sentiment-analysis-pipeline-with-neural-network-kafka-elasticsearch-and-kibana/
[10] Dataset - Twitter US Airline Sentiment https://www.kaggle.com/crowdflower/twitter-airline-sentiment
[11] Dataset - Tweet Sentiment Extraction https://www.kaggle.com/c/tweet-sentiment-extraction/data
[12] Dataset - Preprocessed twitter tweets https://www.kaggle.com/shashank1558/preprocessed-twitter-tweets
[13] Dataset - Apple Twitter sentiment https://data.world/crowdflower/apple-twitter-sentiment/discuss/apple-twitter-sentiment/miztcmjq#mjczxd3y
[14] Dataset - Sentiment 140 http://www.sentiment140.com/

[15] Cornegruta, Savelie & Bakewell, Robert & Withey, Samuel & Montana, Giovanni. (2016). Modelling Radiological Language with Bidirectional Long Short-Term Memory Networks. 17-27. 10.18653/v1/W16-6103.