**Description and Summary**

**Problem Statement:** In our project, we are implementing Count-Min Sketch to retrieve Top-K words from the tweets in real time or a given time frame. The tweets are streamed real-time using tweepy and are sent across a pipeline of Kafka, Logstash, and Elasticsearch. Top-K words are displayed for every 5 minutes which can be dynamically queried with a custom timeframe. Also, the user can query the ElasticSearch index in four different ways to get insights about the tweets.

**Procedure**:

1. Getting real-time data from **Tweepy**: Firstly, we need to stream the tweets in real-time so we use Twitter’s tweepy API and, by creating twitter authentication token and installing tweepy on our system, we can listen to the tweets in real-time.
2. **Pre-processing**: As the data obtained from tweepy contains a lot of information, we need to pre-process it before pushing into our pipeline. We have only taken the tweet and sentiment attribute of the tweet which describes whether the tweet is **{positive, negative, neutral}** in nature. Then the data is filtered to ignore the set of stop-words which are being compiled from the NLTK library and from various other sources, the full list can be found in the dataset section.
3. **Bloom Filters**: We have ignored stop words using the concept of Bloom Filters, which check whether a particular element is present in a set or not. If the Bloom filter predicts that the given word is not in the set then for sure it is not present, but if the filter predicts that if the word in the set then sometimes it might predict wrong which is also called as a false-positive error.

**Implementation**: We have implemented Bloom Filters using **pyprobables** library in python by providing the number of elements to be inserted, and the false-positive rate which can be tolerated. The library then calculates the size of bit-array and number of hash-functions required.

1. **Format of Data**: After ignoring stop words we need to get rid of all the punctuations in the data and emoji’s in the tweets. Then we will tokenize the data, get individual words in the tweet string, and store them into a list. Finally, we convert the data into JSON format in order to push it into Kafka.

**text: “The actual tweet”**

**tokenized: “List of words in the tweet”**

**sentiment: “Positive or negative or neutral”**

1. **Pushing data into Kafka Queue**: We then take our VCL image, start zookeeper and Kafka in it, and create a topic to push the data pre-processed into the Kafka queue.
2. **Installing Logstash and configuring**: Logstash is a plugin based data collection and processing engine. It makes it possible to easily configure it to collect, process and forward data in many different architectures. It acts as a pipeline between Kafka and elasticsearch. Logstash takes input from a Kafka topic and writes this data into elasticsearch index. The details of Kafka topic and port along with elasticsearch index are written in the logstash configuration file. Installing the logstash process is present in the README file.
3. **Installing Elasticsearch and creating an index**: Before running the logstash configuration file we need to install ElasticSearch, the details for which are provided in README. Then we create an index in ElasticSearch to store the data. We execute the Logstash configuration file and verify whether the data is getting populated or not. Also, whenever Logstash pushes data into ElasticSearch, it appends two fields in each document which are the timestamp and the version. So finally, the structure of our document will be as follows:

**@timestamp: “Time at which data is pushed”**

**version: “Document Version”**

**text: “The actual tweet”**

**tokenized: “List of words in the tweet”**

**sentiment: “Positive or negative or neutral”**

1. **Top-k/ Heavy hitters using Count-Min Sketch**: When the user inputs from and to time, our code builds a query and retrieves Top-k words used in tweets.

**Implementation**: We have implemented this using the concept of Count-Min sketch, which stores the occurrences of each word. The count-min sketch data structure is a 2D matrix having rows as the number of hash functions and the number of columns as the size of the bit array. We have used pyprobables library in python to build a Heavy hitter which in-built implements the concept of the count-min sketch. We need to provide the number of top-elements to be queried and the size of the matrix, the library builds the hash-functions and other parameters by itself. We have used 20 hash-functions and size of bit-array as 1000.

1. **Building queries to operate on elasticsearch**: We have implemented the following queries

**Term Query**: When given a term and the timeframe this query retrieves all the tweets in which the term is present.

**Sentiment Query:** When the user selects a particular time frame, this query returns whether the mood of the tweets is either positive, negative and neutral.

**Prefix Query:** When the user selects a particular timeframe and a prefix, the query returns all the tweets in which there are words starting with that prefix.

**Termset Query:** When the user selects a particular time frame, the query returns any tweets that match with at least one or more of the provided terms. The terms are not analyzed, and thus must match exactly. The number of terms that must match should also be entered by the user.

**Summary of Results Obtained**: We tried different approaches including Spark for processing, scripts in elastic search, etc. After extensive analysis, we found this approach to be faster and robust. Our pipeline covers the three phases as mentioned in the project description

Phase 1: We made use of the content shared by the professor, online articles to set up the pipeline and get ourselves educated with the algorithms, streaming the twitter data, basic operations of elastic search, spark, etc.

Phase 2: In this phase, we fixed on our document structure and performed load testing to compare the approaches. We also developed the queries and implemented the pipeline from Vue.js, Flask, Kafka, Logstash to Elasticsearch.

Phase 3: We read the previous papers of Count-Min and Bloom filters and implemented them in our workflow. We performed extensive pre-processing of data and also did the Count-Min Sketch.