**EMOTION CLASSIFICATION - AN ARCHITECTURAL VIEW**

1. **INTRODUCTION**

Emotion classification plays a vital role in determining the emotion quotient of the people on various topics. Social media has been a place for the people to express their feelings about various situations of events happening across the world. Understanding the trends in the emotions help us in coming up with various conclusions in making very important decisions. Twitter is one of the central places where people tend to express their feelings about specific events. Understanding the emotions of the people based on the tweets can help us conclude on various decisions. This project is about building an architecture to handle the tweets from various sources like Twitter API’s and other historical storages like csv’s, json’s etc. This can help us in launching various projects immediately and visualizing the emotions of people quickly.

We have developed an architecture of a pipeline involving various steps like data collection, preprocessing, classifying emotions and visualizing the emotions. The classification of emotions is done using NRC Lexicon. A lightweight tool is used to handle the search capability of emotions based on the various keywords and visualizing the same. We have also built a configuration framework where we can configure different data sources and launch the projects with a few button-clicks and see the visualizations instantly on the visualization tool.

This document is composed of sections explaining various tools used, the architecture of the pipeline for handling the data from various sources and, classifying the emotion and visualizing the same.

1. **TOOLS**
   1. **Kafka:**

Kafka is a distributed streaming platform. In simple terms it acts like a queue to handle the high scale of incoming data. When we are streaming the data from high scale platforms, it is necessary to get the data flowing as fast as possible. It will not be feasible to process the data synchronously which is when the Kafka becomes a handy tool to keep the data streaming. Kafka will act like a central broker to store the data in the intermediate state and the consumers associated with the broker will process the messages. This type of architecture helps in processing the messages parallelly using multiple consumers retrieving the messages from the broker. We will be using twitter API to stream the tweets and the scale at which the tweets will be streamed is very high an hence the Kafka helps us in handling this type of task.

* 1. **Spark:**

Spark is a general-purpose computing framework. It helps us in processing huge volumes of data in a distributed manner. Spark gives us the capability of processing the streams of data. We are making use of this feature of spark to process the high streams of data using the spark structured streaming capabilities.

* 1. **Elasticsearch:**

Elasticsearch is an open source search and analytics engine. It helps us in storing the unstructured data and provide the search and analytics capabilities. We are going to use Elasticsearch at various steps of our pipeline and store the outputs.

* 1. **Kibana:**

Kibana is an open source tool belonging to the elastic stack. It helps us in building the visualizations of the data stored in the Elasticsearch. We can quickly visualize the data in various formats. It also gives us an interface to quickly search using simple queries on the data stored in the Elasticsearch.

* 1. **Django:**

Django is an open source web development framework to build the web applications. We are going to use the Django to quickly launch the web interface to build a framework for configuring various data sources and projects and launch them seamlessly.

1. **HIGH LEVEL VIEW**

At a very high level, the architecture consists of two types of data sources, an online stream of data which could be from twitter, news feeds, forums etc., and any historical static storage which could be anything ranging from file systems like csv’s, json’s, xml’s to databases and data warehouses.

**A screenshot of a cell phone

Description automatically generated**

As shown in the above image, once the data is collected from the any of the sources, it will be passed through the step of preprocessing and feature engineering where we clean, process and perform the feature engineering which will further be used to train the machine learning models. Once the machine learning model is trained and tested, it will further be deployed for inference and will be scored on the real-time data. These steps which are included in the box are iterative and will be repeated in regular intervals of time based on the scoring of the models.

The current project is completely based on the NRC-Lexicon to classify the emotions. Hence all the steps included in the box are replaced by the NRC-Lexicon.

1. **HISTORICAL DATA**

The historical data generally means the static data which is already collected and stored and will be processed using the technique of batch processing. The data will be uploaded into a common storage location which will be tied to a spark file streaming engine. Any newly uploaded file will be picked up by the spark streaming processor. The spark processor will perform two operations, one to dump the raw data into the raw Elasticsearch index and the other to classify the emotions using the NRC lexicon and stores the emotions in the new Elasticsearch index. This helps us to have the raw data in our infrastructure and backtrack for any future purposes. The file offline\_classifier.py is used to perform these operations. The figure below shows the clear view of the pipeline.

**A close up of text on a white background

Description automatically generated**

1. **STREAMING DATA**

The real-time data is streamed from twitter using the twitter’s streaming API. We are using the tweepy package to stream the tweets. The process streaming\_service.py process is used to stream the tweets. The tweets will be instantly pushed to the Kafka broker specific to the project. The messages from the Kafka broker will be retrieved by online\_loader.py which will dump the raw data into the Elasticsearch index and an other process online\_classifier.py which will classify the emotions and stores them in the other Elasticsearch index.

**A screenshot of a cell phone

Description automatically generated**

1. **HANDLING NEW PROJECTS**

This project provides a framework to configure new data sources using the entry points like **offline loaders** and **twitter tracks** in the admin website. Each of the data source is tied to a specific job. A job is a process which runs based on the configuration specified. Once the data sources are created, one can launch a new project from the **projects.** Once a new project is created and saved, all the jobs tied to the data sources will be launched and the visualizations can be created from the kibana. The image below show the class diagram of the models which are used to handle the new projects.

**A close up of text on a black background

Description automatically generated**

1. **VISUALIZATION**

Once the emotions are loaded into the Elasticsearch, we can launch Kibana for building the visualizations. The Kibana can be configured with any of the indexes available in the Elasticsearch. The following are the sample visualizations and dashboards created using the elections tweets.

**A screenshot of a social media post

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

1. **CONCLUSION**

This is the architecture which will be highly beneficial for the purpose of creating the visualizations of the emotions and decisions very quickly. The pipeline helps in handling different sources like streams of data from twitter and static data sources as well. We leveraged the benefits of distributed streaming and structured streaming of Kafka and Spark respectively to achieve real-time view of the emotion index of the people. This can further be improved to include the machine learning module to train, deploy and score the performance of the models in the real-time environment and iterate based on the performance.