

## **Real-Time Emotion Detection Using Kafka and Machine Learning**

Ann Maria John, Divya Neelamegam, Kartik Mukkavilli, Poojitha Venkat Ram, Shruti  
Badrinarayanan

Department of Applied Data Science, San Jose State University

DATA 228: Big Data Technologies and Applications

Guannan Liu, Ph.D

December 6, 2023

## **Abstract**

Real-time emotion detection has become a crucial technical undertaking in an era where comprehending human emotions is essential for several applications, including customer experience enhancement, sentiment analysis, and tailored services. To recognize and assess human emotions in real-time, this project offers an integrated system that makes use of Apache Kafka, Spark Streaming, and machine learning (ML). A comprehensive emotion detection dataset from Kaggle is used for the project, this dataset is centered around English Twitter messages with the goal of categorizing emotions. It contains tweets labeled with six primary emotions: anger, fear, joy, love, sadness, and surprise. Furthermore, they've gathered Twitter data with distinct hashtags corresponding to eight emotions. In total, the dataset files, including both the downloaded and generated data, occupy approximately 8.11 MB of storage. This dataset serves as a valuable asset for our research involving emotion recognition and sentiment analysis, particularly within the Twitter context. The machine learning model will be trained and evaluated on this steady stream of textual data to deliver immediate feedback on the emotional context of the input text. For the system to process the incoming real-time stream of data, the trained model is incorporated into a Spark Streaming application. Seamless communication between the system's producer and consumer components is enabled by the use of Apache Kafka which acts as the main messaging system. Users can enter textual statements into the producer component, which is implemented in Python, and those statements are immediately sent to the Kafka server. The Kafka-integrated consumer component receives this incoming textual data stream and launches the trained ML model in the Spark Streaming application for immediate analysis. The ML model analyzes the textual data and identifies the emotional

undertone to produce instant feedback. We aim to use classification models like Naive Bayes, Support Vector Machines and Random Forest. Fine-tuning of the system architecture ensures high throughput and low latency to handle the constant flow of data, and the former grows to accommodate large-scale data streams by utilizing Spark Streaming's parallel processing capabilities. Potential use cases of the proposed project include sentiment analysis in social media, customer feedback analysis, individualized marketing campaigns, and adaptive user interfaces that take into account user moods.

## **Problem Statement**

Real-time comprehension and response to human emotions have become essential in today's digital world for several businesses, including marketing, healthcare, and entertainment. The difficulty, though, is in effectively gathering, analyzing, and deciphering the enormous volumes of emotional data produced by many sources, such as social media, consumer contacts, or live events. Current techniques frequently fall short of real-time, accurate, and scalable emotional analysis. Thus, a system that combines strong machine learning models with Kafka, a distributed streaming platform, is required to enable real-time emotion analysis and detection from continuous streams of data. For this system to provide immediate insights into human emotions and enable organizations and apps to successfully respond to emotional cues, it must address challenges with data ingestion, processing speed, model accuracy, and scalability.

## Introduction

In the contemporary landscape, understanding and interpreting human emotions in real-time has become imperative for various applications such as refining customer experiences, sentiment analysis, and personalized services. This project addresses the challenge of real-time emotion detection by presenting an integrated system harnessing the capabilities of Apache Kafka, Spark Streaming, and machine learning (ML). The foundation of this endeavour is a robust emotion detection dataset sourced from Kaggle, primarily centred around English Twitter messages. The dataset, comprising tweets annotated with six primary emotions—anger, fear, joy, love, sadness, and surprise—provides a rich resource for training and evaluating the ML model.

The primary objective of the project is to develop a system capable of immediately recognizing and assessing the emotional context of textual input. This involves training and deploying machine learning models, including classification algorithms such as Naive Bayes, Support Vector Machines, and Random Forest, on a steady stream of Twitter data. To seamlessly handle the real-time flow of information, the trained model is integrated into a Spark Streaming application. The communication between the system's producer and consumer components is facilitated by Apache Kafka, which acts as the central messaging system. The project focuses on ensuring high throughput and low latency through fine-tuning the system architecture, leveraging Spark Streaming's parallel processing capabilities to effectively accommodate large-scale data streams.

The potential applications of this project extend to diverse scenarios, including sentiment analysis in social media, analysis of customer feedback, personalized marketing campaigns, and

adaptive user interfaces that dynamically respond to user moods. The integration of Apache Kafka, Spark Streaming, and machine learning algorithms for real-time emotion detection systems with broad implications for enhancing human-computer interactions and applications across various domains.

## **Literature Review**

The literature surrounding real-time emotion detection from textual data has witnessed a surge in interest due to its broad applications in areas such as customer experience enhancement, sentiment analysis, and personalized services. In addressing the challenges of real-time emotion recognition, researchers have delved into various methodologies.

Machine learning techniques have been extensively applied in prior studies for emotion detection, with sentiment analysis serving as a foundational framework for understanding sentiment polarity, a crucial component in emotion detection (*Pang & Lee, 2008*). The integration of Apache Spark Streaming in emotion detection systems has been explored for its efficacy in processing continuous data streams with low latency, aligning seamlessly with the project's goal of delivering immediate feedback on emotional context (*Zaharia et al., 2012*).

In the realm of real-time applications, the adoption of Apache Kafka as a messaging system has gained prominence, leveraging its distributed and fault-tolerant nature for seamless communication between system components—a critical aspect for handling the constant flow of textual data in emotion detection (*Kreps et al., 2011*). The utilization of Twitter datasets similar to the one described in the project has been prevalent in previous works, underscoring the richness of social media data for training and evaluating emotion recognition models (*Mohammad et al., 2018*).

The applications of real-time emotion detection extend across various domains, including social media sentiment analysis, customer feedback analysis, and the development of adaptive user interfaces that dynamically respond to user moods (*Bollen et al., 2011; Liu, 2015; Picard, 1997*).

The recurrent theme in literature emphasizes fine-tuning system architecture for high throughput and low latency in handling large-scale data streams, with Spark Streaming's parallel processing capabilities aligning effectively with this objective (*Carbone et al., 2015*).

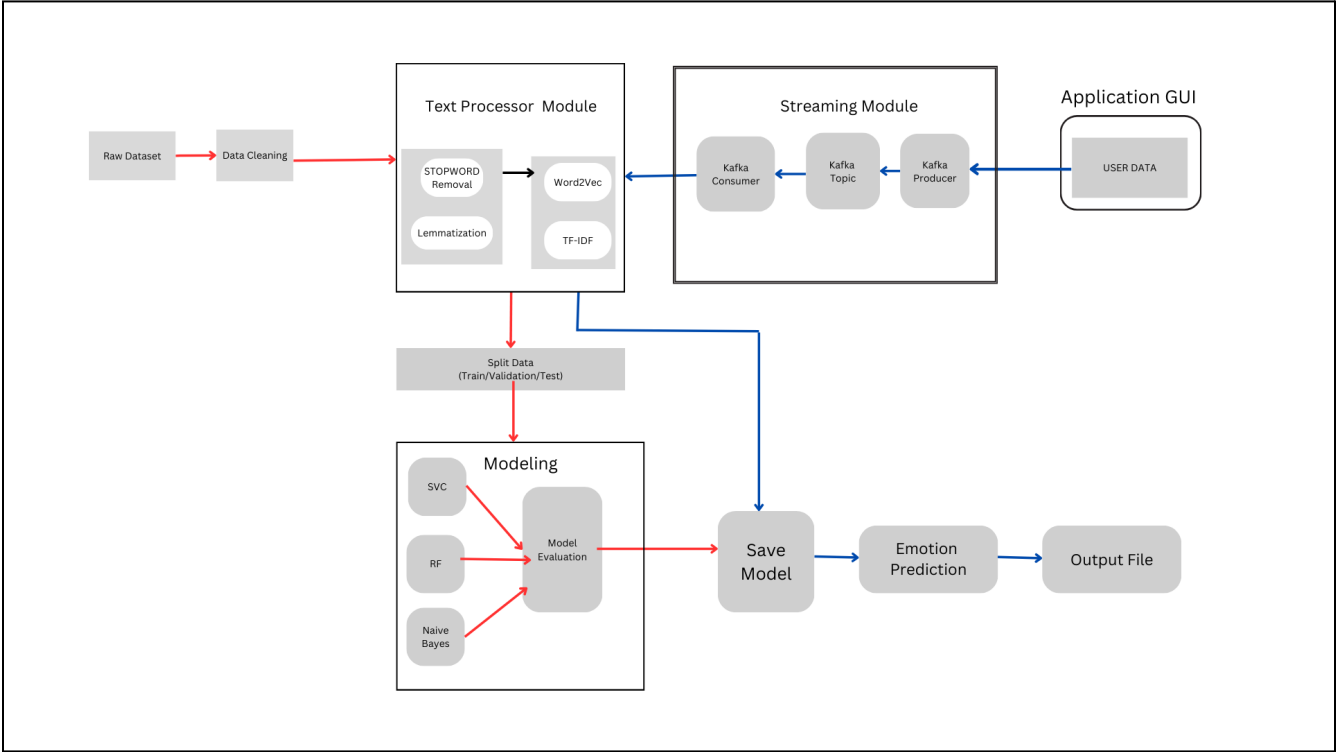
This comprehensive literature review establishes the groundwork for the project, demonstrating the relevance of machine learning, real-time processing, and effective system architecture in advancing the field of emotion detection from textual data.

## **Methodology**

Figure 1 presents an overarching methodology for a predictive analytics system, which processes text data for emotion prediction. The process flow begins with the raw dataset undergoing initial data cleaning to strip out irrelevant information and prepare the data for nuanced processing. This cleaned data is then fed into the Text Processor Module, where it is further refined through techniques such as stopwords removal, lemmatization, and vectorization through Word2Vec and TF-IDF methods to enhance the semantic and contextual understanding of the text. Subsequently, the refined data is divided into separate subsets for training, validation, and testing, ensuring that the model can be trained effectively, validated for accuracy, and tested for performance. Various classification models including Support Vector Classifier (SVC), Random Forest (RF), and Naive Bayes are employed within the Modeling stage, each providing a different approach to the predictive task. After these models are trained and evaluated, the best-performing model is saved. The system then utilizes this saved model within

its Streaming Module, which integrates with an application GUI for real-time user interaction for getting input. Data from the application is streamed and processed in real-time, enabling the system to make emotional predictions. The output of these predictions is then exported to an output file, which can be used for further analysis or as a direct feedback mechanism within the application. This end-to-end system is designed to be both scalable and responsive, catering to the needs of dynamic real-time data analysis and prediction.

**Figure 1. Methodology Workflow**



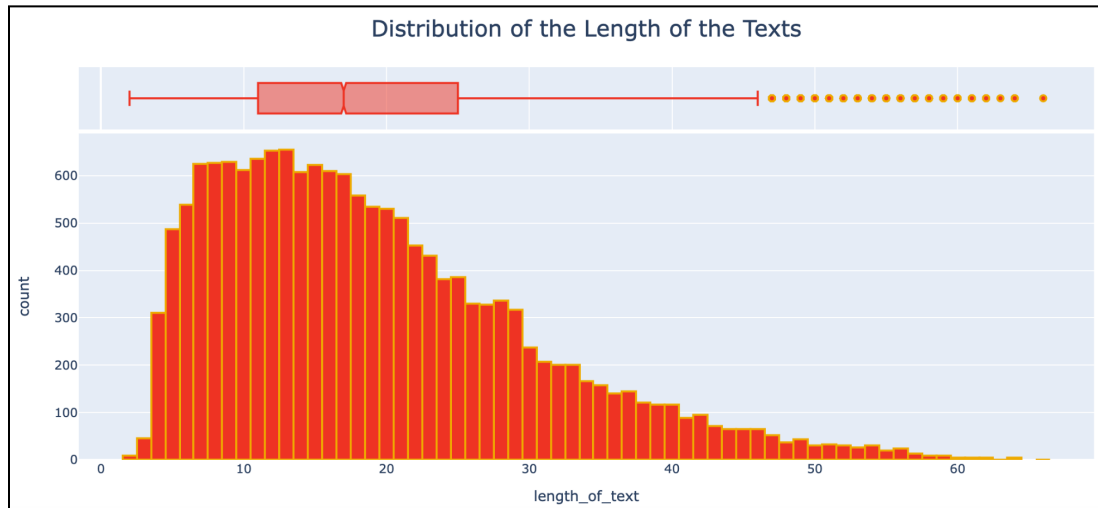
**Exploratory Data Analysis**

Exploratory data analysis was performed on the training data set. There were 16000 rows of which one was a duplicate row. That row ended up being removed from the dataset. A histogram shown in Figure 2 was created using Plotly to understand the distribution the the length of all the sentences in the text column. The median number of words in a sentence was



17. The fewest number of words was 2 while the most was 66. Outlier values ranged from 47 to 66.

**Figure 2.** Distribution of the Length of the Texts



A treemap as seen in Figure 3 was also created to show the top 200 most common words in all the sentences. The word 'i' was found to be the most common word with 25,858 appearances with 'feel' coming in second place with 11,182 appearances.

**Figure 3.** Frequency of the Words in the Train Dataset

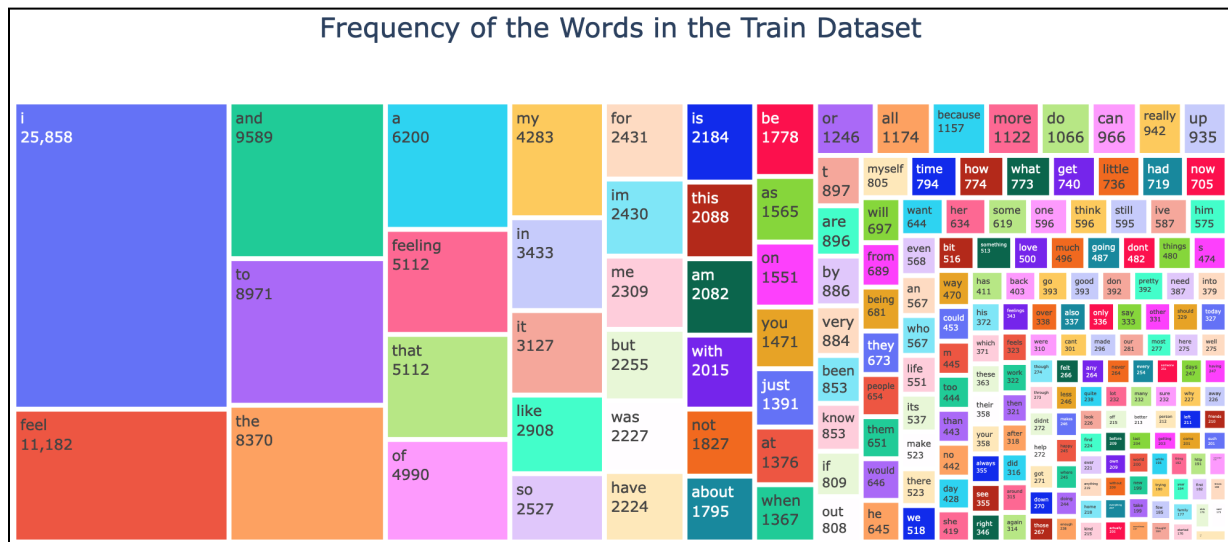
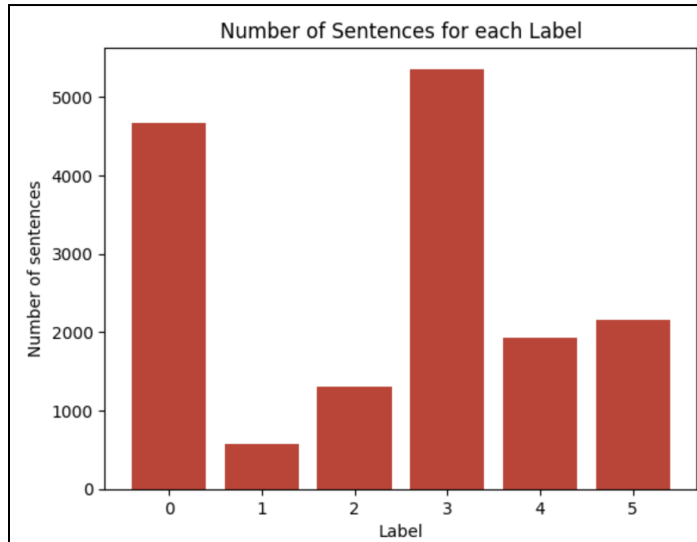


Figure 4 shows a bar graph created to show how many sentences are assigned for each label (0, 1, 2, 3, 4, and 5). Labels 0 and 3 had the most sentences while 1 had the fewest sentences.

**Figure 4.** Number of Sentences for Each Label



### Data Source and Description

The dataset was chosen from [Kaggle](https://www.kaggle.com/datasets/maecap/emojis). It is a collection of English Twitter messages designed for emotion recognition tasks. It is structured to reflect six basic emotions: anger, fear, joy, love, sadness, and surprise. The tweets were collected using the Twitter API by the authors, with a set of hashtags corresponding to eight basic emotions, adding anticipation and trust to the aforementioned six.

The data consists of two features: “text” and “label”. The text is a tweet collected with the label being the corresponding emotion, the target variable.

The data has already been split as training, validation and test sets by the authors which the team has downloaded as CSV files from Kaggle. We used this data as-is for the preprocessing stage. The raw data samples are shown in Figure 5-7.

**Figure 5. Raw Data Sample of Training Set**

	text	label
0	i didnt feel humiliated	0
1	i can go from feeling so hopeless to so damned...	0
2	im grabbing a minute to post i feel greedy wrong	3
3	i am ever feeling nostalgic about the fireplac...	2
4	i am feeling grouchy	3
...	...	...
15995	i just had a very brief time in the beanbag an...	0
15996	i am now turning and i feel pathetic that i am...	0
15997	i feel strong and good overall	1
15998	i feel like this was such a rude comment and i...	3
15999	i know a lot but i feel so stupid because i ca...	0

16000 rows × 2 columns

**Figure 6. Raw Data Sample of Validation Set**

	text	label
0	im feeling quite sad and sorry for myself but ...	0
1	i feel like i am still looking at a blank canv...	0
2	i feel like a faithful servant	2
3	i am just feeling cranky and blue	3
4	i can have for a treat or if i am feeling festive	1
...	...	...
1995	im having ssa examination tomorrow in the morn...	0
1996	i constantly worry about their fight against n...	1
1997	i feel its important to share this info for th...	1
1998	i truly feel that if you are passionate enough...	1
1999	i feel like i just wanna buy any cute make up ...	1

2000 rows × 2 columns

**Figure 7. Raw Data Sample of Test Set**

	text	label
0	im feeling rather rotten so im not very ambiti...	0
1	im updating my blog because i feel shitty	0
2	i never make her separate from me because i do...	0
3	i left with my bouquet of red and yellow tulip...	1
4	i was feeling a little vain when i did this one	0
...	...	...
1995	i just keep feeling like someone is being unki...	3
1996	im feeling a little cranky negative after this...	3
1997	i feel that i am useful to my people and that ...	1
1998	im feeling more comfortable with derby i feel ...	1
1999	i feel all weird when i have to meet w people ...	4

2000 rows × 2 columns

## Data Preprocessing

This section outlines the data preprocessing steps taken on the textual data. The primary objective of these steps was to clean and transform the raw text data into a format that is suitable for training machine learning models.

Here is an Overview of the Preprocessing Steps:

1. **Lowercasing Text:** The text data is first converted to lowercase to ensure uniformity, as text data often contains a mix of uppercase and lowercase letters which can be treated differently by models. **Tokenization:** The text is split into individual tokens (words) using NLTK's `word_tokenize` method. Tokenization is essential for processing individual words in the text.
2. **Handling Negations and Lemmatization:** **Negations:** Words following "not" are prefixed with "not\_" to preserve their negation context, which is crucial for understanding the sentiment or meaning of sentences.
3. **Lemmatization:** Each word is lemmatized using NLTK's `WordNetLemmatizer` to convert it to its base or dictionary form. This process reduces the complexity of the text data by combining different forms of the same word.
4. **Removing Stop Words and Non-Alphanumeric Characters:** Stop words (commonly used words that carry little semantic meaning, like 'the', 'is', etc.) are removed using `ENGLISH_STOP_WORDS` from `scikit-learn`. Additionally, punctuation and numbers are removed to focus only on textual data.
5. **TF-IDF Vectorization:** The text data is transformed using the `TfidfVectorizer` from `scikit-learn`. The vectoriser is configured to include unigrams, bigrams, and trigrams

(ngram\_range=(1, 3)) and to ignore words that appear in less than two documents (min\_df=2). TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure used to evaluate the importance of a word to a document in a collection of documents. It helps in understanding the context and relevance of words in documents.

The preprocessing function is applied to training, validation, and test datasets. After preprocessing, the text data in these datasets are vectorized using the TF-IDF vectorizer. The vectorized data is saved in a sparse matrix format as .npz files to save space as shown in Figure 8. These files are stored in a directory named 'Preprocessed Data'. The paths to these saved files are provided for reference and later use in the machine learning pipeline.

**Figure 8.** Preprocessed Data saved in Sparse Matrix Format

```
('Preprocessed Data/training_tfidf.npz',
 'Preprocessed Data/validation_tfidf.npz',
 'Preprocessed Data/test_tfidf.npz')
```

The preprocessing steps applied in this project are crucial for cleaning and transforming raw text data into a useful format for machine learning models. By converting text into a numerical form through TF-IDF vectorization, and ensuring that the data is cleaned and standardized, the quality and performance of the subsequent machine learning models are expected to be enhanced. The features for final preprocessed data can be seen in Figure 9 below.

**Figure 9.** Features for Preprocessed Data

```
array(['aa', 'aa meeting', 'abandon', ..., 'zoom', 'zooming', 'zumba'
      dtype=object)
```

## **Model Description**

### **Naive Bayes**

The real-time emotion detection model incorporates a Naive Bayes implementation as one of its machine learning components. Trained on a comprehensive emotion detection dataset, the Naive Bayes model serves as the cornerstone of the system's classification capabilities, leveraging the probabilistic principles of Bayes' theorem for effective analysis of emotions in textual data.

Integrated into the application, the Naive Bayes model, pre-trained on the emotion detection dataset, facilitates the analysis of incoming textual data. This integration ensures a seamless and real-time process, allowing for instantaneous feedback on the emotional context of the input.

During the prediction phase, the Naive Bayes model computes the probabilities associated with each emotion category based on the observed features within the text. The category with the highest probability is then assigned as the predicted emotion. The incorporation of Naive Bayes underscores the model's efficacy in swiftly and accurately classifying emotions in diverse textual inputs, showcasing its technical prowess in the context of real-time emotion detection.

### **Support Vector Classifier**

The real-time emotion detection model also integrates a Support Vector Classifier (SVC) as a vital element within its machine learning framework. SVC is a supervised learning model widely used in classification because of its versatility in handling sparse data. The model was trained meticulously on the emotion detection dataset. The SVC excels at drawing complex borders by utilizing kernel functions and margin maximization concepts. This allows for the accurate identification of emotions inherent in textual data.

SVC uses its trained model to categorize new data points during the prediction phase. Finding support vectors, essential for determining the decision boundary, is the first step in this process. A kernel function is applied to convert data into a higher-dimensional space for more effective separation. Based on a new data point's position concerning the border and the margins a class label is assigned to the former. The SVC is an effective technique for predictive modeling because it can reliably categorize unknown data by utilizing learned decision boundaries and support vectors.

### **Random Forest Classifier**

The Random Forest Classifier (RFC) is another essential machine learning component in our real-time emotion detection algorithm. Using ensemble learning and decision trees to its advantage, this classifier is highly effective at analyzing the subtleties of textual data and provides detailed and comprehensive evaluations of emotions. Its power comes from combining predictions from multiple different trees, each of which adds a unique viewpoint, therefore enabling the system to identify emotions broadly and flexibly.

RFC uses an ensemble of decision trees to classify new data points during the prediction process. Every decision tree assesses the incoming data point's properties on its own and makes a forecast. The RFC aggregates these individual tree predictions via a group decision-making process using a voting mechanism.

## Model Comparison and Results

### Naive Bayes

#### *Baseline Model*

The model achieved an overall accuracy of 73.9%, indicating its ability to correctly classify instances across multiple classes. The precision and recall values vary across classes, with Class 2 exhibiting perfect precision but a low recall, emphasizing potential challenges in identifying instances of this class. The macro-average F1-score is 0.55, reflecting a moderate balance between precision and recall on average across all classes. The weighted average F1-score is 0.69, indicating an overall reasonable performance. The confusion matrix provides a detailed breakdown of the model's predictions, highlighting areas of strength and weakness. Class 0 and Class 1 show relatively strong performance, while Class 2, 3, and 5 demonstrate specific challenges, as seen in the precision-recall trade-offs. These results serve as a baseline for model evaluation and suggest potential areas for improvement in future iterations of the classifier.

#### *Naive Bayes and Hyperparameter Tuning*

The Multinomial Naive Bayes classifier, fine-tuned with hyperparameters {'alpha': 0.5, 'fit\_prior': False} as shown in Table 2 through GridSearchCV, showcases strong performance in both validation and test sets. The accuracy of 83.8% on the validation set and 82.6% on the test set attests to the model's ability to generalize well to unseen data. The confusion matrix and classification report reveal high precision, recall, and F1-score values across various classes, suggesting effective discrimination among categories. The weighted average F1-score of 0.84 on the validation set and 0.82 on the test set underlines the model's overall balanced performance.



These results underscore the success of the hyperparameter tuning process in enhancing the Multinomial Naive Bayes model's classification accuracy and robustness.

The model was retrained using the optimal hyperparameters and evaluated on both the validation and test sets. After demonstrating its effectiveness, the completed Naive Bayes model was saved as a joblib file (naive\_bayes\_model.joblib) for easy deployment and usage at a later time.

**Table 1:** Best Hyperparameters for Naive Bayes

Parameter	Parameter Value
alpha	0.5
fit_prior	False

### Support Vector Classifier

Using the training dataset, a baseline SVC model was constructed, and predictions were then made using the validation set. This baseline model's performance was assessed thoroughly using common classification measures, including accuracy, precision, recall, F1-score, and confusion matrix. Hyperparameter tuning using grid search cross-validation (a 5-fold cross-validation approach) was done to find the best hyperparameters (Table 2).

**Table 2:** Best hyperparameters for Support Vector Classifier

Parameter	Parameter Value
C	1
kernel	linear

The model was retrained using the optimal hyperparameters and evaluated on both the validation and test sets. After demonstrating its effectiveness, the completed SVC model was

saved for live data prediction as a joblib file (svc\_model.joblib) for easy deployment and usage at a later time.

### Random Forest Classifier

A baseline model using the training dataset was established and its predictions were evaluated on the validation set. Classification metrics including precision, recall, F1-score, accuracy, and confusion matrix were employed to assess the beeline model's performance. Hyperparameter tuning using grid search cross-validation with a fold of 5 was conducted. The model was retrained using the optimal hyperparameters obtained in Table 3.

**Table 3:** Best hyperparameters for random forest classifier

Parameter	Parameter Value
criterion	gini
max_depth	30
min_samples_leaf	10
n_estimators	150

The robustness and generalizability of the trained model was then evaluated by evaluating its performance on the validation and test sets. The completed model was saved as a joblib file for easy deployment and usage at a later time.

A comparison of the model's performance metrics shows that the SVC performs the best on several different criteria. SVC outperforms RFC at 56.3% validation and 57.6% test accuracy and the Naive Bayes classifier at 83.8% validation and 82.5% test accuracy, with a robust validation accuracy of 88% and matching test accuracy. SVC demonstrates a well-rounded performance as evidenced by its 88% performance in precision, recall, and F1-score. RFC trails behind with an

accuracy of 79%, recall of 58%, and an F1-score of 56%, while the Naive Bayes model closely tracks SVC, scoring highly in precision and F1-score at 82% and recall at 83%. Overall SVC emerges as the most effective model for this dataset. Table 4 shows a model performance comparison.

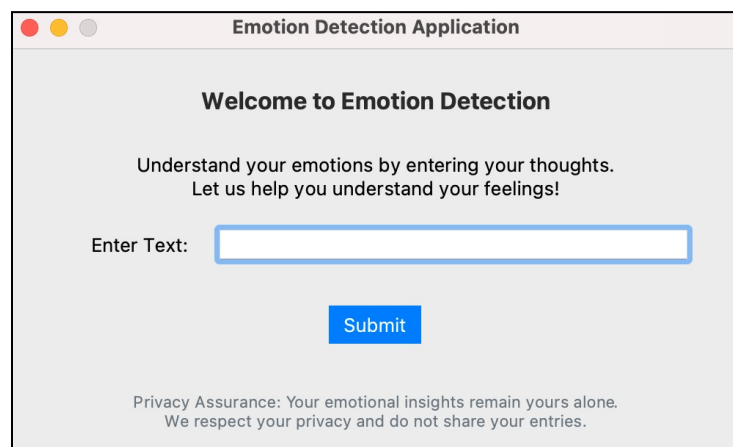
**Table 4:** Model performance comparison

<b>Evaluation Metrics</b>	<b>Naive Bayes</b>	<b>RF</b>	<b>SVC</b>
<b>Validation Accuracy</b>	83.8%	56.3%	88.0%
<b>Test Accuracy</b>	82.5%	57.6%	88.0%
<b>Precision</b>	82.0%	79.0%	88.0%
<b>Recall</b>	83.0%	58.0%	88.0%
<b>F1 Score</b>	82.0%	56.0%	88.0%

### **Emotion Detection Real-Time GUI Application**

#### **PYQT GUI Application**

The Emotion Detection Application showcased in Figure 10 is a tool developed using PyQt and Python, which are popular choices for crafting interactive and responsive graphical user interfaces. PyQt, a set of Python bindings for the Qt application framework, provides the necessary tools to create a visually appealing and user-friendly interface. This particular application benefits from PyQt's capabilities, offering an intuitive layout that invites users to input text for real-time emotion analysis. Python's versatility and its extensive library ecosystem allow for the seamless integration of complex backend algorithms that can process natural language and assess emotional content.

**Figure 10.** Application Window

This UI serves as the gateway for accepting live data from users, and behind the scenes, the application employs text analysis and machine learning algorithms to interpret the emotional sentiments expressed in the user inputs. Once a user enters their text into the designated area and clicks the 'Submit' button, the data is processed using a pre-trained model to predict the underlying emotions. The emphasis on privacy in the GUI reassures users that their data is handled securely, which is particularly crucial when dealing with the sensitive nature of personal emotional expressions. This application exemplifies how Python, together with PyQt, can be harnessed to create powerful applications that are both accessible and secure for end-users.

### **Apache ZooKeeper**

Apache ZooKeeper complements Apache Kafka by providing distributed coordination and synchronization services. In the real-time emotion detection project, ZooKeeper is employed to manage the distributed nature of the Kafka cluster, ensuring synchronization and consistent configuration across all nodes.

ZooKeeper is particularly crucial for maintaining the configuration details, such as the health and status of Kafka brokers, partitions, and other cluster metadata. This coordination ensures that the Kafka consumers can reliably retrieve and process messages from the correct partitions, contributing to the overall fault tolerance and reliability of the real-time data processing pipeline.

### Apache Kafka Topic

The real-time emotion detection project is based on Apache Kafka, which functions as a distributed streaming platform. It is designed to handle large-scale data streams and provides a publish-subscribe messaging system. Kafka excels in processing and managing real-time data, making it an ideal choice for applications requiring low-latency data pipelines.

For Real Time emotion detection, Kafka is utilized to facilitate the seamless flow of live streaming input data from Emotion Detection GUI Application. A dedicated Kafka topic, named "emotion-detection-stream," as shown in Figure 11, has been created to serve as the communication channel for these messages. This Kafka topic is configured with a replication factor of 1 and a single partition, ensuring simplicity and ease of management. Figure 12 provides a comprehensive snapshot, listing the Kafka topics on the server. This visual representation allows for a quick assessment of the current state of topics, facilitating streamlined monitoring and administration of the real-time data communication infrastructure.

**Figure 11.** Establishing 'emotion-detection-stream' Kafka Topic for Message Communication

```
(base) guruprasanthmuthu@Guruprasanth's-MacBook-Pro kafka_2.12-3.6.0 %  
(base) guruprasanthmuthu@Guruprasanth's-MacBook-Pro kafka_2.12-3.6.0 % bin/kafka-topics.sh --create --bootstrap-server localhost:9092  
--replication-factor 1 --partitions 1 --topic emotion-detection-stream
```

**Figure 12.** Overview of Kafka Topics on the Server

```
(base) guruprasanthmuthu@Guruprasanth's-MacBook-Pro kafka_2.12-3.6.0 % sh bin/kafka-topics.sh --list --bootstrap-server localhost:9092  
emotion-detection-stream  
(base) guruprasanthmuthu@Guruprasanth's-MacBook-Pro kafka_2.12-3.6.0 %
```

## Kafka Producer

To handle the posting of user input messages to the Kafka topic, a Kafka producer is employed within the Emotion Detection GUI Application. This producer is responsible for sending the user's input messages to the specified Kafka topic. The simplicity of the Kafka producer enables efficient communication between the GUI Application and the Kafka cluster. The 'Emotion\_kafka\_producer' class, as illustrated in Figure 13, has been crafted to seamlessly bridge the communication between the Emotion Detection GUI Application and the Kafka cluster. The producer is instantiated with the specified bootstrap servers, connecting it to the Kafka broker at 'localhost:9092'. Utilizing the dumps function from the json module, the producer serializes messages into a JSON format before transmission, enhancing compatibility and ease of consumption by downstream components.

Error handling is a pivotal aspect of the producer's design. In the event of a Kafka-related error during initialization or message sending, the class gracefully handles exceptions, printing informative error messages for diagnosis. Exception types such as `KafkaTimeoutError` and generic `KafkaError` are caught, allowing for differentiated responses based on the nature of the error.

The send method is the core functionality of the producer, responsible for sending messages to the 'emotion-detection-stream' Kafka topic. Each message is structured as a dictionary with a 'key' set to `None` and the 'value' containing the actual message. This design choice ensures simplicity in message handling and aligns with the conventions of Kafka's key-value pairs.

**Figure 13.** Code Implementation of the Emotion\_kafka\_producer Class

```

6 class Emotion_kafka_producer:
7     def __init__(self):
8         try:
9             self.producer = KafkaProducer(
10                 bootstrap_servers=['localhost:9092'],
11                 value_serializer=lambda x: dumps(x).encode('utf-8')
12             )
13         except KafkaError as e:
14             print(f"Error initializing Kafka producer: {e}")
15             # Handle the exception, e.g., log the error or raise it to the calling code
16
17     def send(self, message):
18         try:
19             data = {'key': None, 'value': message}
20             print("Sending message to Kafka producer", data)
21             future = self.producer.send('emotion-detection-stream', data)
22             # Ensure the message is sent successfully before continuing
23             record_metadata = future.get(timeout=10)
24             print("Message sent successfully to topic:", record_metadata.topic)
25         except KafkaTimeoutError as e:
26             print(f"Error sending message to Kafka producer: {e}")
27             # Handle the timeout error, e.g., log the error or raise it to the calling code
28         except KafkaError as e:
29             print(f"Error sending message to Kafka producer: {e}")
30             # Handle other Kafka errors, e.g., log the error or raise it to the calling code

```

Upon sending a message, the producer validates its successful transmission by awaiting acknowledgment from the Kafka broker. The get method is employed with a timeout parameter, enabling a graceful exit if acknowledgment is not received within the specified time limit.

### Kafka Consumer

On the consumption side, a Kafka consumer, as depicted in Figure 14, is deployed to continuously listen to the "emotion-detection-stream" topic. The consumer is configured with certain parameters, such as "auto\_offset\_reset" set to 'earliest' to read messages from the beginning, "enable\_auto\_commit" set to False to disable automatic offset commits, and "group\_id" set to None, indicating the absence of a consumer group, thereby ensuring a clean and straightforward data retrieval process. Figure 15 delves deeper into the intricacies of real-time data consumption by the Kafka Consumer. This visual representation encapsulates the

dynamic process wherein live data is efficiently ingested and processed, underscoring the project's commitment to maintaining a responsive and agile infrastructure.

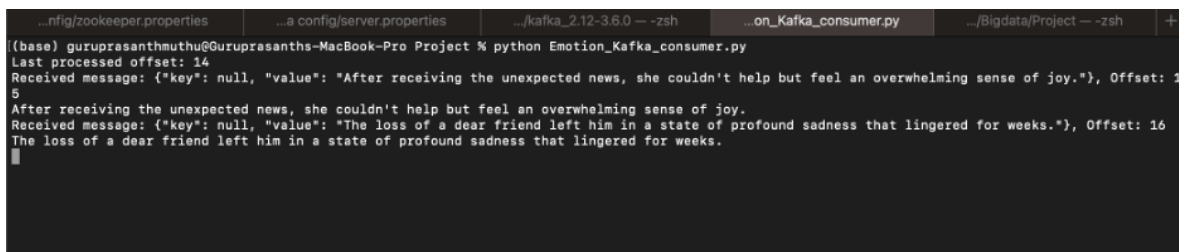
**Figure 14.** Kafka Consumer Configuration

```

4 # Set up the Kafka consumer
5 consumer = KafkaConsumer(
6     'emotion-detection-stream',
7     bootstrap_servers='localhost:9092',|
8     auto_offset_reset='earliest',
9     enable_auto_commit=False, # Disable automatic offset commits
10    group_id=None # Disable consumer group for reading from the beginning
11 )

```

**Figure 15.** Real-Time Data Consumption by Kafka Consumer



```

...nfig/zookeeper.properties  ...a config/server.properties  .../kafka_2.12-3.6.0 -- zsh  ...on_Kafka_consumer.py  .../Bigdata/Project -- zsh  +
(base) guruprasanthmuthu@Guruprasanth-MacBook-Pro Project % python Emotion_Kafka_consumer.py
Last processed offset: 14
Received message: {"key": null, "value": "After receiving the unexpected news, she couldn't help but feel an overwhelming sense of joy."}, Offset: 15
After receiving the unexpected news, she couldn't help but feel an overwhelming sense of joy.
Received message: {"key": null, "value": "The loss of a dear friend left him in a state of profound sadness that lingered for weeks."}, Offset: 16
The loss of a dear friend left him in a state of profound sadness that lingered for weeks.

```

An essential consideration in this architecture is the management of Kafka offsets to prevent data loss in the event of Kafka restarts. To address this, the offset information is stored in an external file named 'external\_offset.txt.' This approach allows the Kafka consumer to maintain the last processed offset, ensuring that, upon restart, it can resume reading from the correct position in the Kafka topic. As the Kafka consumer processes new incremental data, it retrieves messages from the Kafka topic based on the stored offset information. The processed data, which includes messages enclosed in double quotes to handle delimiters (commas) within the data, is then appended to an input file named "tweet.txt." This meticulous formatting, as depicted in Figure 16, is critical for preserving the integrity of the data when later analyzed or utilized in downstream processes. The figure visually presents the structured content of the



"tweets.txt" file, showcasing the precise handling of message delimiters for enhanced data coherence.

The utilization of topics, producers, and consumers, combined with careful offset management, ensures the seamless flow of user input messages and the preservation of data integrity, even in the face of Kafka restarts or system failures. The externalization of offset information and the creation of the "tweet.txt" file contribute to the overall reliability and durability of the data processing workflow.

**Figure 16.** Content of the "tweets.txt" File

```
(base) guruprasanthmuthu@Guruprasanth's-MacBook-Pro:~/Project % cd Tweets_Input
(base) guruprasanthmuthu@Guruprasanth's-MacBook-Pro:~/Tweets_Input % ls
tweets.txt      untitled.txt
(base) guruprasanthmuthu@Guruprasanth's-MacBook-Pro:~/Tweets_Input % cat tweets.txt
"The sun is shining, and the birds are singing; it's a beautiful day"
(base) guruprasanthmuthu@Guruprasanth's-MacBook-Pro:~/Tweets_Input %
```

### Emotion Prediction Using SVC

The evaluation of different machine learning models revealed that Support Vector Classification (SVC) outperformed Random Forest (RF) and Naïve Bayes after meticulous hyperparameter tuning, achieving an impressive accuracy of 88%. Recognizing the significance of real-time emotion detection, the decision was made to deploy the SVC model for live predictions on streaming data sourced from the 'tweet.txt' file.

The streaming data ingestion and processing pipeline kickstarts with the consumption of the 'tweet.txt' file, housing a continuous stream of user-generated text data. The pivotal text processor module is then engaged to preprocess the incoming data comprehensively. The preprocessing steps encompass essential tasks such as stopwords removal, lemmatization, and

vectorization using both Word2Vec and TF-IDF processes. This multifaceted approach ensures that the text data is transformed into a format suitable for effective emotion classification.

The heart of the system lies in the loaded SVC model, finely tuned to capture the nuances of emotion expression within the dataset. As each new batch of data is vectorized, the SVC model, as depicted in Figure 17, is invoked to predict the emotion classification. Leveraging the powerful capabilities of the SVC model, the system achieves an accuracy rate that aligns with the prior evaluation. This streamlined approach to real-time emotion detection ensures both accuracy and efficiency in handling the continuous stream of incoming data.

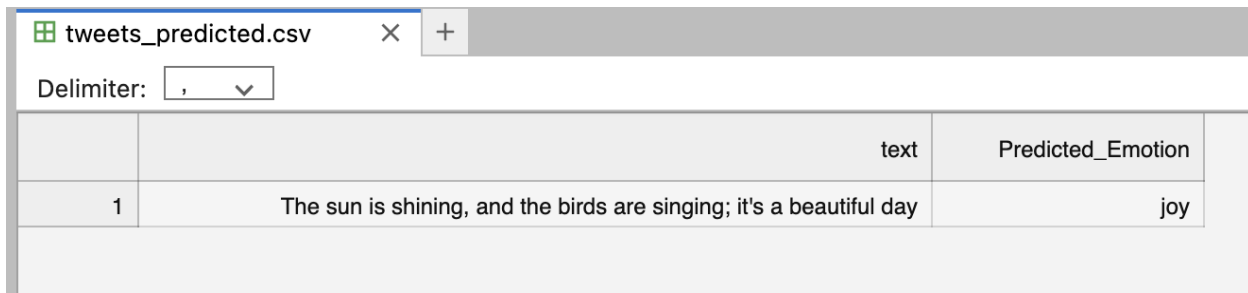
**Figure 17.** Loading the Pre-trained SVC Model for Real-Time Emotion Prediction

```
[179]: # Load the saved model and test accuracy on the test set
import joblib
svc_model = joblib.load('svc_model.joblib')

streaming_data_prediction= svc_model.predict(streaming_data_tfidf)
```

Crucially, the predictions are not merely consigned to the realm of real-time computation; rather, they are systematically cataloged for further analysis and insights. As depicted in Figure 18, showcasing the final predicted emotion of live data, the results encompassing the original tweet and its predicted emotion class are persistently stored in the 'tweets\_predicted.csv' file. This structured storage mechanism not only provides a historical record of emotion predictions but also facilitates downstream analytics for refining and enhancing the model over time. To maintain the integrity of the streaming data and optimize system resources, a post-processing step involves clearing the 'tweet.txt' file after each batch has been successfully processed. This ensures that subsequent data streams are unencumbered by redundant information, streamlining the workflow for ongoing real-time emotion detection.

**Figure 18.** Shows the Final predicted emotion of Live data



	text	Predicted_Emotion
1	The sun is shining, and the birds are singing; it's a beautiful day	joy

In essence, the integration of the SVC model within the real-time emotion detection pipeline, coupled with a robust text processing module, heralds a sophisticated approach to emotion prediction. The harmonious synergy of model performance, preprocessing techniques, and efficient data handling lays the foundation for a dynamic and responsive system capable of discerning and classifying emotions in real-time. As the system continues to evolve, the 'tweets\_predicted.csv' file becomes a valuable repository for ongoing analysis, refinement, and augmentation of the emotion prediction model.

## Conclusion

The integration of machine learning methods, Spark Streaming, and Apache Kafka has enabled the creation of a novel system capable of real-time emotion detection in textual data. This study has resulted in the deployment and comparison of several machine learning models namely, Naive Bayes, SVC, and Random Forest on a stream of Twitter data by utilizing an emotion detection dataset obtained from Kaggle. After a thorough analysis of all of these models, it is clear that SVC is the best model. It has the highest accuracy, precision, recall, and balanced F1-score when measured against the other models. This highlights the effectiveness of SVC in identifying and interpreting emotional contexts in textual inputs, highlighting its potential in a variety of applications such as sentiment analysis, customer feedback analysis, and adaptive

user interfaces. A new age in real-time emotion detection systems is being heralded by the smooth integration of Apache Kafka, Spark Streaming, and SVC. This integration has the potential to transform human-computer interactions and provide many areas with sophisticated insights from textual data.

### **Future Work**

While this project provided insightful information about the application and compared various machine learning models for real-time emotion detection, there are still a few areas that need more research. Future work could focus on multilingual emotion recognition by capturing linguistic nuances using methods such as language-specific fine-tuning or multilingual embeddings. Furthermore, using deep learning models offers a viable method of capturing rich contextual information in textual emotional expressions. BERT and other pre-trained models give the system access to a wealth of information gleaned from enormous text corpora. To sum up, this work serves as a foundation for future research endeavors.

## References

1. Pang, B., & Lee, L. (2008). *Opinion mining and sentiment analysis. Foundations and Trends® in Information Retrieval*, 2(1–2), 1–135.
2. Zaharia, M., et al. (2012). *Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters*. In USENIX Annual Technical Conference.
3. Kreps, J., et al. (2011). *Kafka: A Distributed Messaging System for Log Processing*. In *NetDB, USENIX*.
4. Mohammad, S. M., et al. (2018). *SemEval-2018 Task 1: Affect in Tweets*. In Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018), 1-17.
5. Bollen, J., et al. (2011). *Twitter's mood predicts the stock market*. *Journal of Computational Science*, 2(1), 1-8.
6. Liu, B. (2015). *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. Cambridge University Press.
7. Carbone, P., et al. (2015). *Apache Flink: Stream and Batch Processing in a Single Engine*. *IEEE Data Engineering Bulletin*, 38(4), 28-38.