Hate Speech and Sentiment Analysis of GPT outputs using SparkML

Aashna Kunkolienker, Akshita Upadhyay, Kavya Gupta

Abstract

This project investigates a Big Data pipeline for hate speech detection and sentiment analysis using large-scale distributed computing, Apache SparkML, to improve the safety of LLM systems. The increasing use of Large Language Models (LLMs) has raised significant concerns about content security and the potential spread of harmful or offensive output. Our approach involves generating a dataset using OpenAI's API, followed by preprocessing and the application of various machine learning models within Spark's framework. This allows for the efficient classification of text data with high accuracy Results and model performance are presented with visualizations and pipeline insights.

Introduction

The rapid integration of Large Language Models (LLMs) into various applications has transformed the online landscape, yet at the same time introduced significant challenges in content moderation. Although LLMs offer unparalleled capabilities, they can inadvertently generate harmful, offensive, or biased text. Ensuring the security and safety of LLM-based systems is quite important, particularly on sensitive topics such as social networks, education, and customer service.

To address this, we developed a scalable solution using Apache SparkML for two critical tasks: hate speech detection and sentiment analysis. Unlike traditional machine learning workflows, which often struggle with large datasets, our Big Data approach leverages Spark's distributed processing power to handle massive volumes of text efficiently. By generating a custom dataset using OpenAI's API, we trained models capable of filtering and analyzing text at scale, offering a practical framework for tackling content moderation challenges.

This report details our project's design, including how SparkML pipelines were utilized for preprocessing and model training, the dataset generation process, and the results of our models. Our solution strikes a solid balance between scalability and accuracy, making it a practical option for real-world use.

Big Data Relevance and SparkML Integration

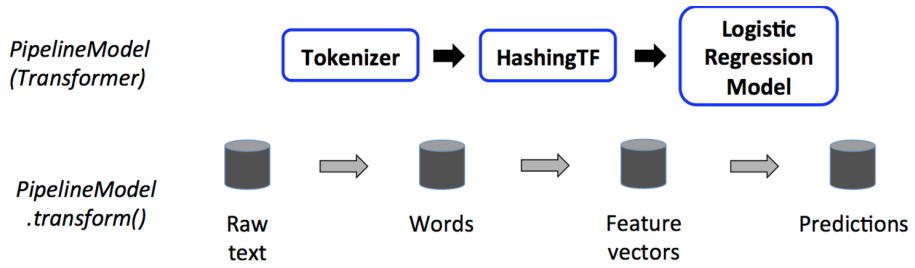
Given the vast amount of data involved in the analysis of text from social networks, traditional computing fails. Spark's distributed processing capabilities are ideally suited to handle such Big Data challenges, enabling efficient processing, reduced processing

times, and improved scalability. We selected SparkML for its seamless integration of machine learning pipelines and it allowed us to streamline feature extraction, model training, and evaluation within a unified framework, making it ideal for our project's needs.

How Spark and SparkML Work (Brief Overview)

Apache Spark is a powerful open-source distributed computing system designed for Big Data processing and analysis. It achieves its speed and efficiency by distributing tasks across a cluster of machines, processing them in parallel, and keeping data in memory whenever possible. SparkML is Spark's machine learning library that builds upon this distributed architecture. This pipeline API simplifies the development of complex machine learning workflows by allowing users to define each stage, from data loading and pre-processing to feature extraction, model selection, and evaluation. This approach makes SparkML particularly well-suited for our project.

Spark Pipeline



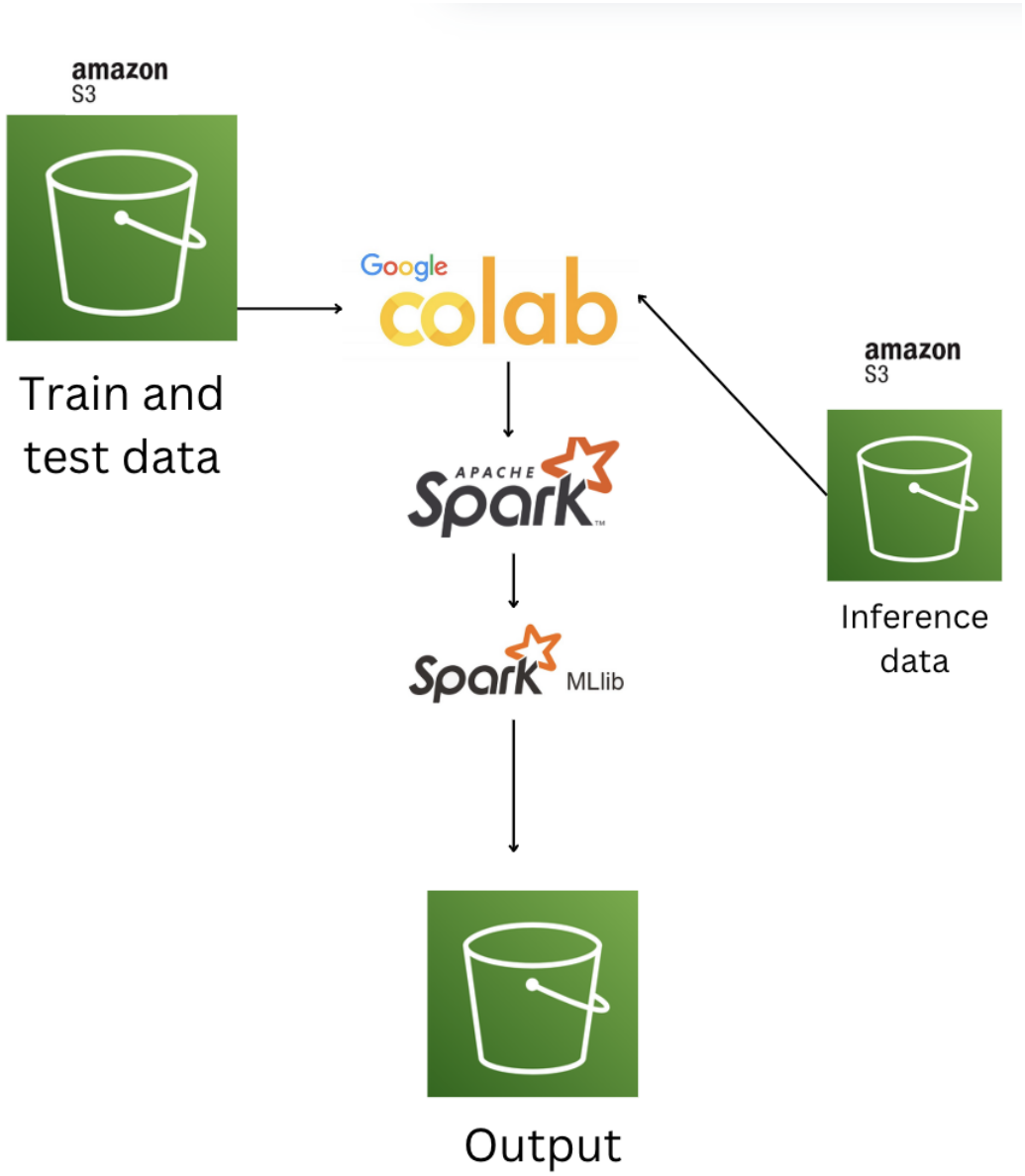
Dataset Description and Technologies Used

The dataset was generated using OpenAI's API to create varied samples of text data containing hate speech and general sentiment. Each sample was meticulously labeled to facilitate supervised learning for both training and testing. The workflow leveraged the following technologies:

- **AWS S3**: Used for storing the dataset and intermediate results during the processing and inference pipeline.
- **Colab Python**: Used for running Python scripts and orchestrating the overall workflow.
- **Apache Spark Preprocessing**: Applied for cleaning and preparing text data, including tokenization, stopword removal, and feature engineering.
- **Spark MLlib**: Used for training machine learning models such as Logistic Regression and Naive Bayes.
- **AWS S3 (Inference Data)**: The preprocessed dataset and model outputs were stored in S3 for further use.

- **Inference in Colab**: The trained models were utilized to make predictions, with the inference logic executed on Colab.
- **AWS S3 (Output Storage)**: The final outputs, including predictions and evaluation metrics, were saved back to S3.

Pipeline Diagram



1 Hate Speech Model

Text Preprocessing and Feature Engineering

Preprocessing the dataset was a crucial step to ensure the text data was clean, tokenized, and transformed into numerical features suitable for machine learning models. We used

Apache Spark for distributed preprocessing to handle large volumes of text data efficiently. The preprocessing steps included:

- **Text Cleaning:** Non-alphanumeric characters were removed from the text using Spark's `regexp_replace` function, and the text was converted to lowercase to ensure consistency.
- **Tokenization:** The cleaned text was tokenized into individual words using Spark's `Tokenizer`, enabling the dataset to be distributed across Spark's processing framework.
- **Stopword Removal:** Common stopwords were removed using Spark's `StopWordsRemover` to reduce noise in the text and focus on meaningful words.
- **Vectorization:** Words were transformed into numerical features using multiple techniques:
  - **TF-IDF:** Term Frequency-Inverse Document Frequency was used to emphasize important terms while reducing the weight of common terms.
  - **N-Grams:** Bigram and trigram features were generated to capture contextual relationships between consecutive words.
  - **Word2Vec:** Pre-trained embeddings were used to represent words as dense vectors, preserving semantic relationships.

The final dataset combined all these features using Spark's `VectorAssembler`, producing a comprehensive feature set for machine learning models. This approach leveraged Spark's parallelism and scalability, ensuring efficient and fast preprocessing of large datasets.

The initial hate speech dataset presented a challenge due to the encoding of the target column, labeled `class`. Upon careful examination, we observed that the values in `class` corresponded to the following: `0` indicated hate speech, `1` indicated offensive language, and `2` indicated neither. To simplify the classification, we redefined the target labels by creating a new label, `class`, where `0` represents "nothing detected" (neither hate speech nor offensive language), and `1` represents "hate speech/offensive language detected."

Using Spark ML

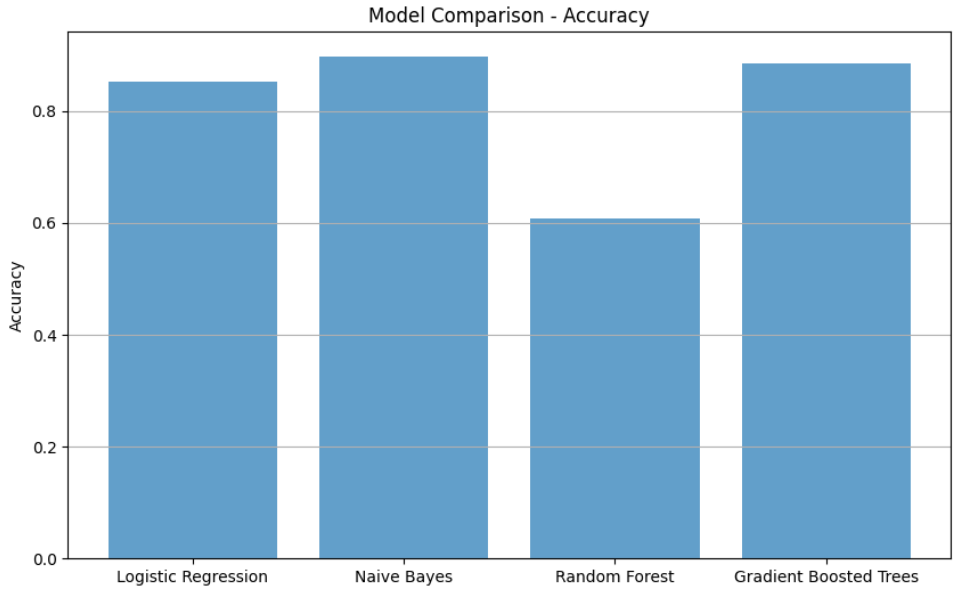
Furthermore, the dataset exhibited a significant class imbalance, with a majority of samples labeled as `class = 1`. To address this, we applied undersampling to the majority class to achieve a balanced dataset. This balanced dataset served as the foundation for training several machine learning models:

- Random Forest
- Logistic Regression
- Naive Bayes
- Gradient Boosted Trees

Gradient boosted trees produced the best results with an accuracy of 90%, which was promising.

Results and Graphs

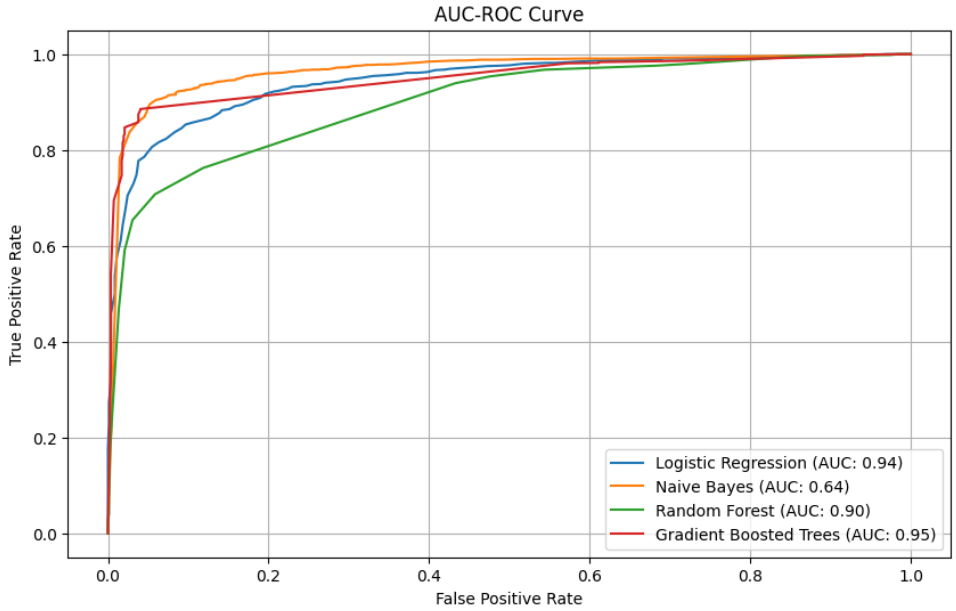
Model Accuracy Comparison



Explanation: The bar chart compares the accuracy of various models:

- **Gradient Boosted Trees:** Achieved the highest accuracy (90%), demonstrating its effectiveness in identifying hate speech and offensive content.
- **Logistic Regression:** Followed closely with accuracy around 89
- **Naive Bayes:** Performed slightly better than Logistic Regression but relies heavily on simplifying assumptions.
- **Random Forest:** Displayed the lowest accuracy (65%), likely due to its limitations in handling the dataset's nuances.

AUC-ROC Curve Analysis



Explanation: The AUC-ROC curves demonstrate the models' ability to distinguish between the two classes:

- **Gradient Boosted Trees:** Showcased the highest AUC (0.95), indicating its superior classification capabilities.
- **Logistic Regression:** Closely followed with an AUC of 0.94, indicating strong predictive performance.
- **Random Forest:** Achieved an AUC of 0.90, reflecting a decent performance.
- **Naive Bayes:** Performed poorly with an AUC of 0.64, suggesting its assumptions may not align with the dataset.

Inference from LLM Predictions

sentence	predicted_class
This product has completely changed my life for the better!	0.0
I usually take a walk every morning to start my day.	0.0
The service at the restaurant was absolutely terrible and disappointing.	0.0
The word 'queer' is often used to describe something unusual or strange in literature.	0.0
Many people in the community proudly identify as queer, embracing their uniqueness.	0.0
I can't stand how some people use the word 'queer' in a derogatory way.	0.0
The customer support team was rude and unhelpful, making me even more frustrated.	1.0
The sun is shining brightly, and it's a perfect day for a picnic.	0.0
In my free time, I enjoy collecting stamps from different countries.	0.0
I strongly dislike how this project has been handled; it's a complete disaster.	1.0
I can stand queers. They are so disgusting!	1.0
Cooking dinner every evening is part of my routine and helps me relax.	0.0
The experience with this airline was the worst I've ever had.	0.0
The term 'queer' has been reclaimed by the LGBTQ+ community as a term of empowerment.	0.0
The vibrant queer community organized a wonderful parade to celebrate diversity.	0.0
Using the word 'queer' as an insult shows a lack of understanding and respect.	1.0
Queer people are ruining everything with their so called rights. They should be cancelled from society.	0.0
It's a warm, sunny day, and the flowers are blooming beautifully.	0.0
I spend hours every weekend painting landscapes to unwind.	1.0
You are a complete idiot and a waste of space	0.0

Explanation: The table presents predictions made by an LLM-based model:

- Most sentences were classified as `class = 0` (non-hate speech), consistent with the LLM's training to avoid generating or promoting harmful content.

- Sentences expressing a lot of negativity or offensive language were flagged as `class = 1`, showcasing the model's capability to identify hate speech.
- Contextual or reclaimed language (e.g., "The vibrant queer community...") was correctly categorized as neutral, highlighting the LLM's semantic understanding.

In summary, Gradient Boosted Trees emerged as the top-performing model, while Logistic Regression also delivered robust results. The LLM-based inference validated the dataset's alignment with ethical AI principles, avoiding over-classification of neutral language.

2 Sentiment Analysis Model

- **Preprocessing:** Text data was cleaned by removing non-alphanumeric characters, converting to lowercase, and eliminating stopwords. Spark's `regexp_replace`, `Tokenizer`, and `StopWordsRemover` functions were used to standardize and tokenize the text across a distributed environment.
- **Feature Engineering:** A combination of Word2Vec embeddings, TF-IDF features, and n-gram representations (bigrams and trigrams) were created. Spark's `Word2Vec`, `HashingTF`, `IDF`, and `NGram` transformers facilitated efficient feature extraction at scale. Features were combined using `VectorAssembler` to create a comprehensive feature set for the models.
- **Model Training:** Logistic Regression and Naive Bayes classifiers were trained using Spark MLlib's distributed machine learning capabilities. The models were trained on the engineered features, allowing for efficient processing of large datasets.
- **Evaluation and Inference:** The models were evaluated using the Multiclass Classification Evaluator with metrics such as accuracy. Spark's distributed computing enabled inference on new datasets, with predictions stored and retrieved seamlessly.

The combination of preprocessing techniques and feature engineering improved the model's accuracy. Specifically, using Word2Vec embeddings, n-grams, and TF-IDF features yielded robust feature representations that contributed to better performance.

Results with Different Feature Combinations

Several combinations of preprocessing and feature engineering techniques were evaluated using Apache Spark to enhance the performance of sentiment classification models. The following results were observed:

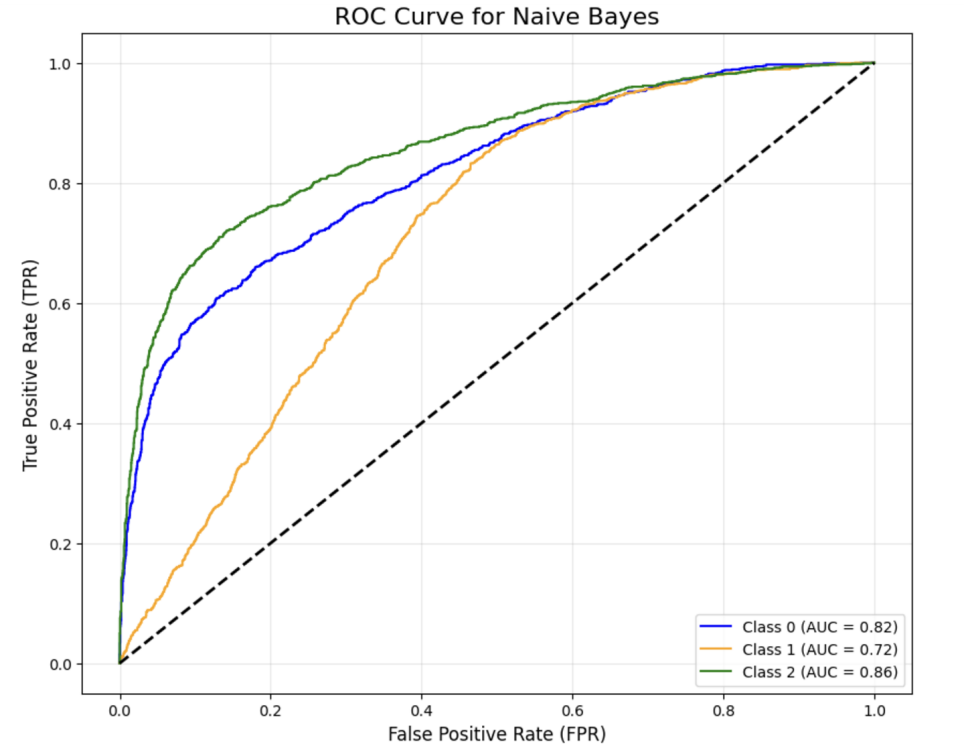
- **Logistic Regression with TF-IDF:** The text was preprocessed using tokenization, stopword removal, and TF-IDF feature extraction. Logistic Regression achieved an accuracy of 55%.
- **Naive Bayes with TF-IDF:** Similar preprocessing steps were applied, followed by training a Naive Bayes classifier. This approach improved accuracy to 58%.

- **Decision Tree with TF-IDF:** The Decision Tree model, trained on TF-IDF features, yielded an accuracy of 50%.
- **Naive Bayes with N-Grams:** Incorporating bigrams and trigrams into the feature set along with Bernoulli Naive Bayes (smoothing = 2.0) achieved the best performance among the evaluated models, with an accuracy of 64%.
- **Logistic Regression with Word2Vec and N-Grams:** Combining Word2Vec embeddings with n-grams as features for Logistic Regression resulted in a lower accuracy of 53%.

These results demonstrate the impact of feature representation and model choice on the performance of sentiment classification. The highest accuracy of 65% was achieved using Naive Bayes with N-Grams.

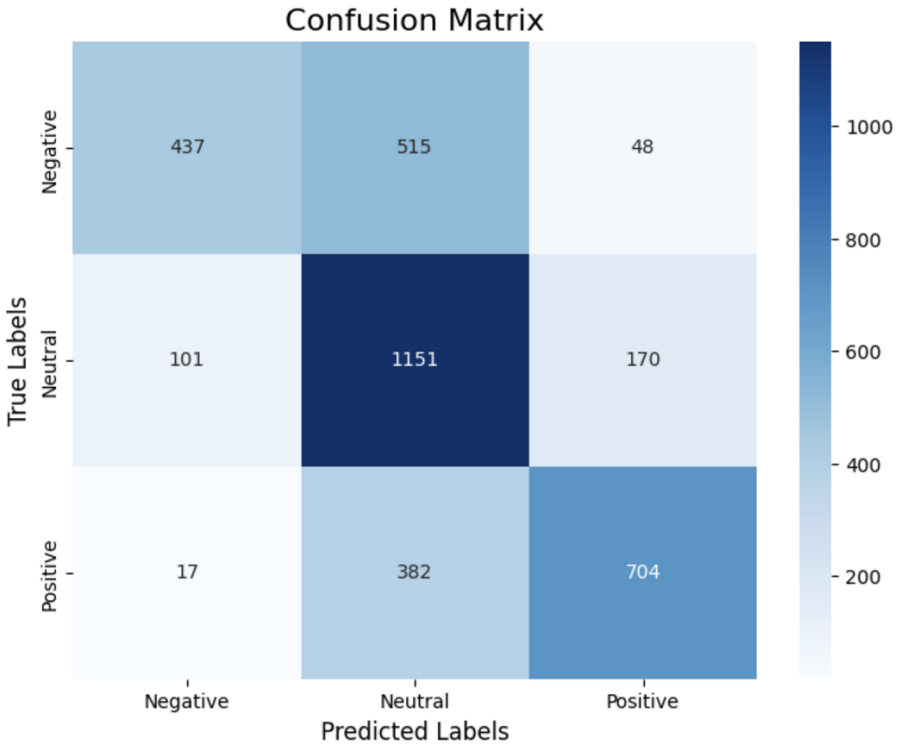
Results and Graphs

Sentiment Analysis AUC



The Area Under the Curve (AUC) for the sentiment analysis model demonstrates its ability to differentiate between positive, neutral, and negative sentiments. A higher AUC value indicates better model performance in distinguishing these classes.

Confusion Matrix for Sentiment Analysis



The confusion matrix illustrates the classification performance of the sentiment analysis model, showing the counts of true positive, true negative, and misclassified instances for each sentiment class. This provides insights into which classes are more prone to misclassification.

Sentiment Inference Results

sentence	prediction
This product has completely changed my life for the better!	2.0
I usually take a walk every morning to start my day.	1.0
The service at the restaurant was absolutely terrible and disappointing.	0.0
The word 'queer' is often used to describe something unusual or strange in literature.	1.0
Many people in the community proudly identify as queer, embracing their uniqueness.	2.0
I can't stand how some people use the word 'queer' in a derogatory way.	1.0
The customer support team was rude and unhelpful, making me even more frustrated.	0.0
The sun is shining brightly, and it's a perfect day for a picnic.	2.0
In my free time, I enjoy collecting stamps from different countries.	2.0
I strongly dislike how this project has been handled; it's a complete disaster.	1.0

This graph presents the sentiment predictions on the inference dataset. The results showcase the model's predictions alongside confidence scores for each sentence, highlighting the distribution of sentiment classifications and areas where the model can be fine-tuned further.

2.1 How We Generated the LLM output Dataset and Performed Inference

Using OpenAI's API, we created a synthetic dataset by generating text samples under various categories, such as hate speech, neutral sentiments, and general opinions. To

achieve this, we wrote a local Python script that utilized the OpenAI API to generate sentences based on predefined prompts. These prompts encompassed a diverse range of scenarios, including positive sentiments, negative experiences, neutral observations, and edge cases designed to test the model's robustness, such as the use of the word "queer" in different contexts.

Due to billing constraints, the dataset was limited to 15 diverse sentences. Each generated sample was programmatically labeled for both hate speech and sentiment classification. Following dataset preparation, we utilized Apache Spark for preprocessing and feature vectorization, enabling efficient distributed processing. Inference was performed using the trained machine learning models to classify the generated text.

2.2 Results

The results of our inference were decent.

The results of our inference showcased the effectiveness of the implemented models. For hate speech detection, the Naive Bayes model achieved a high accuracy of **90%** on the test data, successfully identifying offensive and non-offensive content. Also, very few predictions of the model on the chatGPT outputs were identified as hate speech. This highlights the resilience of the large language model (LLM) used to generate the dataset, which resisted jailbreak attempts even with prompts containing cuss words.

For sentiment analysis, the Naive Bayes model achieved an accuracy of **65%**, reflecting moderate performance on the test data. While the model successfully classified positive, neutral, and negative sentiments in many cases, there is room for improvement. The results on the inference data indicate that the model has potential for fine-tuning, especially with more complex feature engineering or the inclusion of advanced embeddings. Overall, the first iteration of our models demonstrated robust performance for hate speech detection and a solid baseline for sentiment analysis.

section*Challenges Faced and Solutions One of the main challenges faced during the project was achieving better accuracy for the sentiment analysis model. Fine-tuning the model required extensive experimentation with embeddings, tokenization techniques, and hyperparameters. Additionally, feature engineering using embeddings posed difficulties due to the need for thorough dataset understanding and preprocessing complexities. These challenges were addressed by leveraging distributed processing capabilities of Apache Spark, enabling efficient experimentation and feature transformations.

Future Scope

This project can be further extended to incorporate more advanced deep learning models like BERT or GPT for sentiment analysis and hate speech detection. Integrating real-time pipelines for processing streaming data can enhance scalability. Also, exploring multilingual support for text classification would make the system applicable to a broader range of use cases.