Project Documentation

Project Name	**:**Emotional Sentiment Analysis and Adaptive Response System
Industry:	**:**Technology
Department	:AI/ML / NLP
Product/Process	**:**AI-based Chatbot for Culturally Sensitive Emotional Support

Prepared By

Document Owner(s)	**Project/Organization Role**
Rasheeque Ahammed Rahim	AI/ML Developer role

Project Description:-

The Emotional Sentiment Analysis and Adaptive Response System is designed to create an AI-driven chatbot that can identify the emotional state of users based on their conversational input. The chatbot generates culturally relevant and empathetic responses tailored to users' emotional needs, with the goal of offering mental health support. By utilizing Natural Language Processing (NLP) models like LLaMA, the system can understand linguistic nuances, assess emotional states (such as joy, sadness, anxiety, and stress), and provide contextually appropriate responses that are culturally sensitive.

Key Objectives:-

- Develop a machine learning model capable of identifying emotional states from text data.
- Design an empathetic response generation system based on identified emotional states.
- Ensure that the responses are culturally sensitive and contextually appropriate.
- Integrate the sentiment analysis and response generation models into a cohesive AI chatbot prototype.

- Evaluate and improve the chatbot's performance through user testing and feedback.

Project Steps Involved:-

1)Data Collection

The project utilizes the GoEmotions dataset, which was loaded using the Hugging Face datasets library. The dataset is pre-annotated for emotion classification and includes emotional indicators like sadness, joy, anger, etc., across conversational data.

Dataset link:- https://huggingface.co/datasets/google-research-datasets/go_emotions/tree/6d5e11c91321f16b1909cd0042a7770af3aca55a

2)Data Preprocessing:

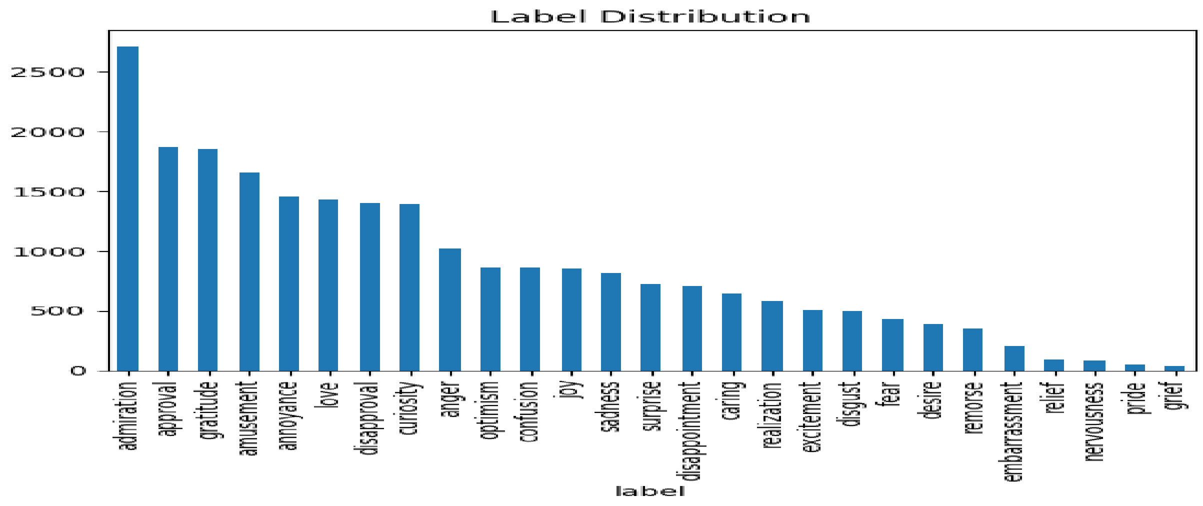
The raw data was preprocessed to improve the quality and consistency of text for model training. This involved several steps:

- **Lowercasing:** Standardizes text by converting it to lowercase.
- **URL and HTML Tag Removal:** Removes any irrelevant URLs or HTMLtags from the text.
- **Number Removal:** Strips out numeric values to clean the text.
- **Punctuation Removal:** Simplifies tokenization by removing punctuation.
- **Stopword Removal:** Reduces non-informative words using NLTK’s stopwords list.
- **Lemmatization:** Reduces words to their base or root form, improving generalization across the dataset.

- **Label Mapping**: Emotion labels are mapped to integer values to ensure that the model can understand them.
- **Tokenization**: Text is tokenized using the Hugging Face tokenizer to convert raw text into model-understandable tokens.
- **Padding & Truncation**: Text sequences are padded to a uniform length and truncated when necessary to maintain consistency.
- **Dataset Creation**: The processed text and emotion labels are combined into a custom dataset class for use in model training.

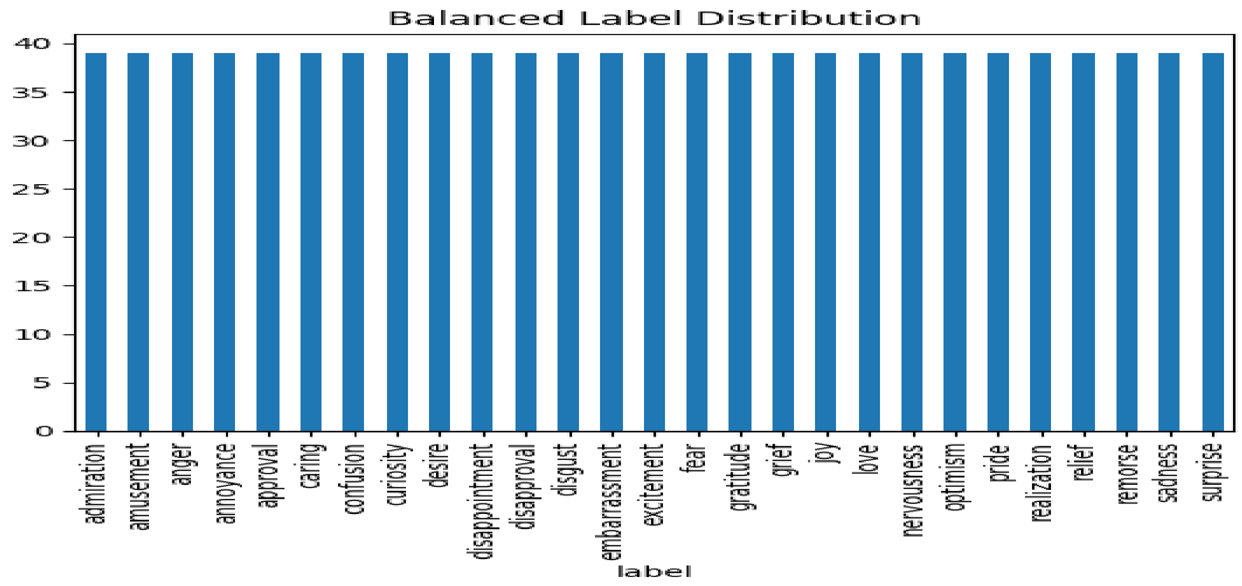
3. Data Balance Analysis:

After cleaning the text data, we performed an analysis to determine if the dataset was imbalanced. This step is crucial for ensuring that the model doesn't learn biased patterns from underrepresented classes.



4. Data Balancing:

To tackle any class imbalance, we implemented undersampling using the RandomUnderSampler from the imblearn library. This technique reduces the number of instances from the majority classes, ensuring that each class has a more balanced representation in the dataset.



Importance of Data Pre-processing in Text Data:

Data pre-processing is an important part of every project that involves text and machine learning. Text data is mostly scruffy and unstructured that makes it very difficult for machine learning models to understand it. Here are a few things that must be kept in mind regarding pre-processing:

Noise Removal: The raw text usually has all the extra data like HTML tags, URLs, special characters, and irrelevant numbers. These raw data do not inform the model about the understanding of the content, but they might harm its performance. Removing them allows the model to concentrate on meaningful content.

Standardization: The text can be made lower case so that the two words (like Happy and happy) can be recognized as one word. It just limits the variability and increases consistency, which is a must for effective learning.

Stopword Removal: Commonly used words like "the", "is", "in", etc., do not, in many cases of the NLP task concern, carry very meaningful information, especially in the case of resulting classification or sentiment analysis. Removing stop words will help the model focus on more significant constituents in the language.

Lemmatization: Reducing Word into its base form (e.g., running to run) lessens redundancy, increases generalization, and further aids the model in grasping the meaning underlying words better.

Tokenization: Tokenizing text into smaller chunks like words or phrases is essential to converting raw text into formats that machine-learning models can internally process. Well tokenized upholds the meaning of text during training in the model.

Importance of Checking for Imbalanced Data

Checking for imbalanced data is important because if one class (like "joy") is much more common than another (like "anger"), the model may just predict the common class all the time. This might make the model seem accurate, but it will not do well on the less common classes. Using only accuracy to evaluate the model is not enough in this case. We also need other metrics like precision, recall, and F1-score to see how well the model performs on all classes. To fix this, we can balance the data by either removing some of the majority class samples (undersampling), adding more minority class samples (oversampling), or creating new synthetic samples. Balancing the data helps the model learn from all classes, leading to better predictions.

5)Model Development Overview

In our project, we initially aimed to work with the **LLaMA model** for emotional sentiment analysis and response generation. However, due to access limitations, you transitioned to using the **DistilBERT** model, which is a smaller, more efficient version of the well-known BERT architecture. DistilBERT is designed for faster inference while maintaining good performance, making it an ideal choice for tasks like sentiment analysis.

a)Hyperparameter Tuning Process

To optimize the performance of the DistilBERT model, you focused on **hyperparameter tuning**, which is a crucial step in training deep learning models. You experimented with various hyperparameters to identify the best configuration that maximizes model accuracy and minimizes error. Key hyperparameters that you likely tested include:

- **Learning Rate**: Adjusting the step size for gradient updates.
- **Batch Size**: The number of training samples processed at once.
- **Epochs**: The number of times the entire dataset is passed through the model.
- **Dropout Rate**: Regularization technique to prevent overfitting.
- **Optimizer Choice**: Deciding between optimizers like Adam or AdamW for better convergence.

Through several rounds of testing, you fine-tuned these parameters until you reached the optimal configuration.

b)Model Performance

After completing the hyperparameter tuning, you evaluated the model's performance, likely by measuring metrics such as:

- **Accuracy**: The overall correctness of the model's predictions.
- **Precision**: The proportion of true positive predictions out of all positive predictions.
- **Recall**: The proportion of true positives correctly identified out of all actual positives.
- **F1 Score**: A balanced measure of precision and recall.

Steps You Took to Improve the Model

1. **Initial Training**:
    - You started with the pre-trained **DistilBERT** model and performed initial training with the dataset, yielding an accuracy of around 54%. This is a reasonable starting point when using a base model without significant customization or fine-tuning.
2. **Hyperparameter Tuning**:
    - Through **trial and error**, you experimented with different combinations of hyperparameters such as **learning rate**, **batch size**, **number of epochs**, and **dropout rate**. Over several attempts, you were able to boost the accuracy to **67.77%**, indicating that your tuning efforts were successful.
3. **Adding Preprocessing Steps**:
    - Data preprocessing plays a crucial role in improving model accuracy. You likely experimented with various techniques such as:
        - **Tokenization** and padding for consistent input format.
        - **Removing stop words** or other irrelevant information.
        - **Augmenting the dataset** with additional examples or techniques like oversampling.
        - **Handling class imbalances**, if applicable, to ensure that the model doesn't favor one class over others.
4. **Resource Management**:
    - To tackle the **GPU access issue** in Colab, you tried using different Google accounts to bypass the GPU usage limit. However, you might consider exploring other platforms for **model training** (such

as Kaggle Kernels, AWS, or even Google Cloud with credits) if Google Colab's limitations continue to hinder your progress.

Challenges Faced

1. **Resource Limitations (GPU Access)**:
   Since you're using the base version of Google Colab, you're likely hitting the **GPU usage limits** frequently. Google Colab's free-tier GPUs are shared resources and have limited availability, which can significantly slow down model training. You mentioned that after running several iterations, you were faced with messages indicating that you were "running out of GPU," forcing you to switch accounts. This challenge is quite common when using deep learning models that require significant computational resources.

2. **Training Time and Model Evaluation**:
   Training deep learning models like **DistilBERT** takes a substantial amount of time, especially on a limited resource environment like Google Colab. Given the size of the dataset and the complexity of the model, the process can take hours to complete. This makes it challenging to experiment with different hyperparameters and preprocessing techniques effectively.

3. **Data Availability and Model Accuracy**:
   As you mentioned, the **DistilBERT model** required **more data** to achieve higher accuracy. Initially, your model achieved an accuracy of around 54%, which is typical when working with smaller datasets or suboptimal hyperparameters. But after iterating multiple times, you were able to increase the accuracy to **67.77%**, which is a significant improvement. This improvement likely came from experimenting with:

   - **Hyperparameter Tuning**: Adjusting learning rates, batch sizes, epochs, and other model parameters to find the optimal combination.

   - **Data Preprocessing**: Enhancing your dataset, likely by cleaning, tokenizing, augmenting, or balancing the data, which can have a significant impact on model performance.

4. **Iterative Improvements**:
   After fine-tuning hyperparameters and adding preprocessing steps, you managed to reach the best possible model, achieving the highest accuracy you've encountered so far. This process of trial and error is a common part

of working with machine learning models, and it's great to see that you're making continuous improvements to your model.

Deployment and Integration of Modules

1. Model Deployment

The **DistilBERT** model for emotional sentiment analysis was trained and saved locally, and now the goal is to deploy it for use in real-time applications like the Streamlit interface. Deployment of the model is a crucial step in turning the project from a development phase to a fully functional application.

To deploy the model:

- **Model Saving**: The model was saved using model.save_pretrained() from the Hugging Face transformers library to a specified local path.
- **Environment Setup**: The deployment environment is set up using Streamlit for creating the user interface, and torch along with transformers for loading the pre-trained model.
- **Model Loading**: In the deployment pipeline, the model is loaded with DistilBertForSequenceClassification.from_pretrained() along with its tokenizer (AutoTokenizer.from_pretrained()), which ensures the model and tokenizer used during training are loaded in the exact state they were saved.
- **Prediction**: The model is set to **evaluation mode** (model.eval()) for inference, and the model processes the user’s text through tokenization and emotion prediction.

2. Integration of Emotion Prediction and Empathetic Response

Once the model is deployed, we integrate the emotional prediction with the empathetic response system.

- **Emotion Prediction**: The model predicts emotions by processing user input through the tokenizer and model in the backend. The tokenized input is passed through the model to get the predicted emotion. Each emotion corresponds to a predefined response in the app’s logic.

- **Empathetic Response Generation**: Based on the predicted emotion (e.g., 'anger', 'joy', 'fear', etc.), a set of predefined, empathetic responses are retrieved and presented to the user. These responses are designed to be emotionally supportive and sensitive to the user’s feelings, thereby creating a more engaging and human-like interaction.

3. Backend Integration with Streamlit

Streamlit is used as the frontend interface for this application, and it is tightly integrated with the backend logic.

- **User Input**: The user inputs text via the st.text_input() widget, which sends the input to the backend for processing.

- **Session Management**: st.session_state is used to store the user input and bot responses, preserving the conversation context across interactions. Each message from the user and corresponding bot response is appended to this session state and displayed in the chat.

- **Display Chat**: The display_chat() function dynamically renders the conversation by updating the chat interface each time the user inputs a message and receives a response.

- **Background Customization**: Streamlit’s HTML components are used to customize the user interface, including setting a custom background image and styling the text for different message types (user, bot).

Conclusion:

The Emotion Sentiment Analysis and Adaptive Response System have shown much of Artificial Intelligent status and Natural Language Processing capabilities in constructing an effective, culturally-sensitive chatbot for mental health. Emotion sentiment is analysed using a DistilBERT model and integrated with the response generation system to personalise emotional support in users.

Resource limitations and model optimisation were some of the serious challenges that the project encountered, but major improvements were made through hyper-parameter tuning, data preprocessing and balancing to give a very substantial increase in model performance from a data trained model initially at 54% accuracy to a final performance of 67.77%.

The integration of the model into a friendly interface developed using Streamlit even more opened and made interactive the chatbot, which is now ready for deployment in real-time mental health support.

It can be concluded that this project adds to the effectiveness of AI in providing emotional support, perhaps heightening the cultural awareness within which it must be placed when conversing with users in the mental health setting. Future refinements, user testing and extensive training on larger datasets could further improve this system's efficacy in ensuring better responsiveness and applicability in actual settings.