**Assignment Documentation**

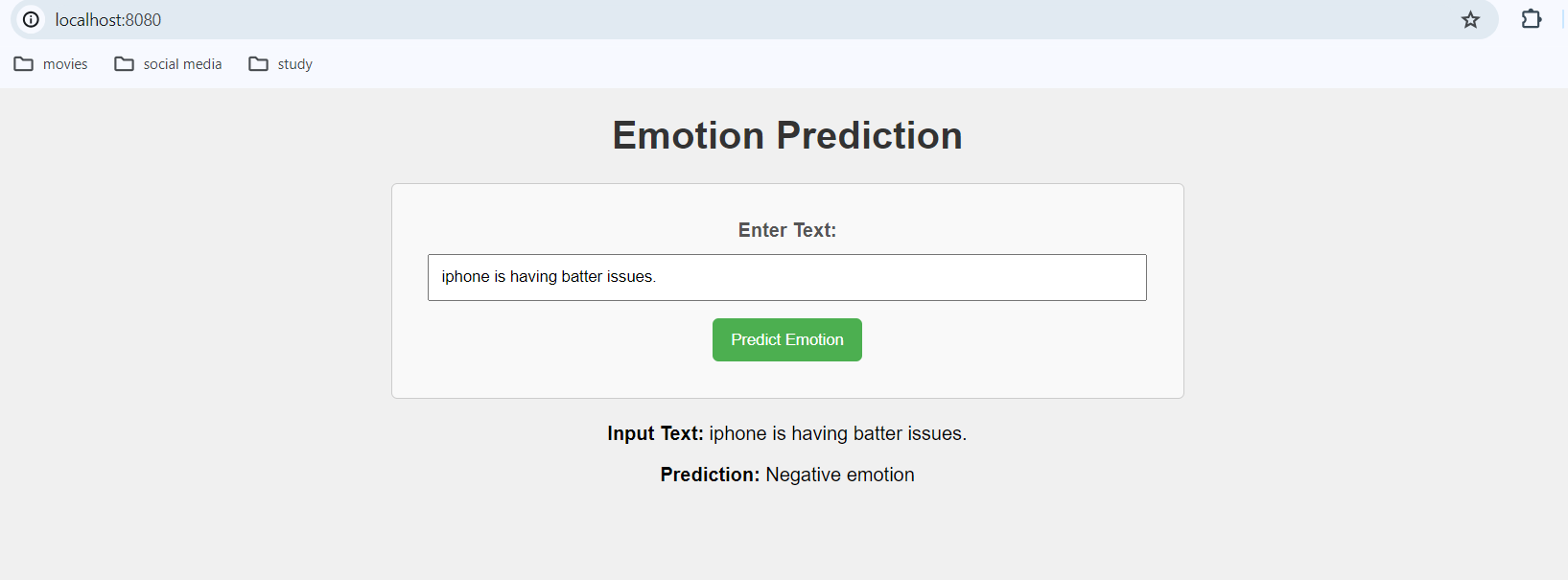
**Problem:** What emotion is directed towards which product

**Technology :** Python, BERT, LLM, HTML, CSS, JavaScript, FastApi

**Website video:** [Click here](https://www.loom.com/share/29ee393fb96b47fbad589fc7d07edc23?sid=371b2838-3007-48ca-9072-fb118ae68681)

**GitHub Repo:** [click here](https://github.com/mandarwarghade/wysa-)

**Snapshot:**

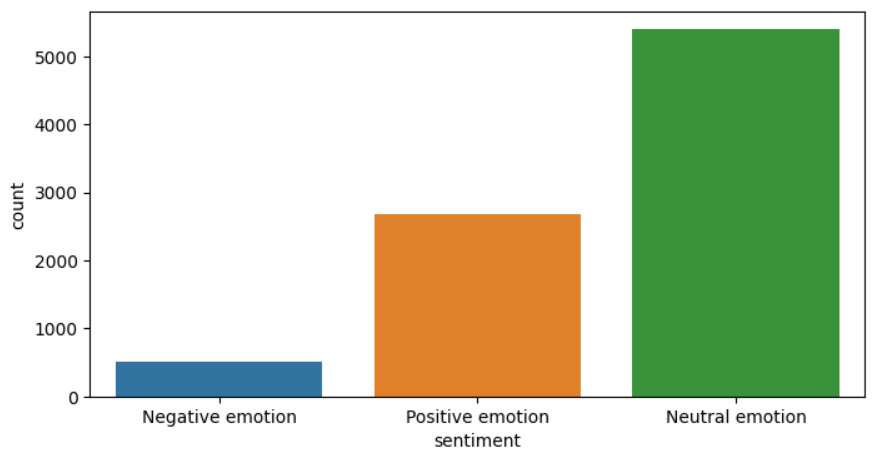
****

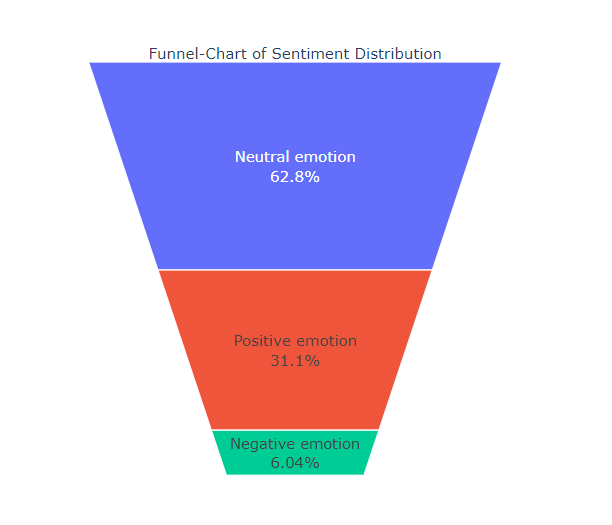
**Exploratory Data Analysis (EDA):**

1. **Overview of Data:** The dataset consists of two sets: a training set with 8,588 tweets and a test set with 504 tweets.
2. **Data Information:** The training set contains three columns: 'tweet\_text,' 'emotion,' and 'sentiment.' The 'tweet\_text' column has 8,588 non-null entries, 'emotion' has 3,291 non-null entries, and 'sentiment' has 8,588 non-null entries.
3. **Handling Missing Values:** The 'tweet\_text' column had one missing value, and the 'emotion' column had 5,298 missing values. The missing value in 'tweet\_text' was addressed by dropping the corresponding row.
4. **Column Renaming and Sentiment Conversion:** Columns were renamed to improve clarity: 'emotion' and 'sentiment.' Sentiments 'No emotion toward brand or product' and "I can't tell" were converted to 'Neutral emotion.'
5. **Sentiment Distribution:** The sentiment distribution in the training set is as follows:

| **Sentiment** | **Tweet Count** |
| --- | --- |
| Neutral emotion | 5,397 |
| Positive emotion | 2,672 |
| Negative emotion | 519 |

**Some charts:**

****

****

****

1. **Text Cleaning** Text cleaning functions were applied to the 'tweet\_text' column:

* Conversion to lowercase.
* Removal of square brackets, links, punctuation, and words containing numbers
* Text within square brackets is removed.
* Symbols and special characters are removed from the tweet text using regular expressions
* Removal of stop words.
* Removal of symbols and special characters.

1. **Conclusion** The text cleaning process ensures that the tweet text is standardized and ready for further analysis. It aids in reducing noise and irrelevant information, allowing for more accurate sentiment analysis and model training. The cleaned corpus is now well-prepared for subsequent natural language processing tasks.

**Approach 1:**

* **FINE TUNED WITH TRANSFORMER MODEL BERT**

1. **Label Encoding:** The code initializes a LabelEncoder to encode the 'sentiment' column in the 'train' dataset. The original text labels are transformed into numerical values.
2. **Importing Libraries and Model Initialization:** The code imports necessary libraries for numerical operations, model evaluation, and PyTorch. It also initializes the BERT Tokenizer and Sequence Classification model with three output labels.
3. **Data Preprocessing:** The code extracts tweet text and sentiment labels from the 'train' dataset, splits the data into training and validation sets using a stratified approach, and tokenizes the tweet text using the BERT Tokenizer with padding and truncation.
4. **Torch Dataset Creation:** The code defines a custom dataset class for PyTorch, accommodating tokenized encodings and optional labels. It implements methods for item retrieval and dataset length.
5. **Model Training and Evaluation** The code defines training and validation datasets, computes evaluation metrics, and initializes a Trainer for model training and evaluation. It trains the model and prints evaluation metrics.
6. **Fine-Tuning Rationale** The purpose of fine-tuning BERT is to adapt a pre-trained model to a specific task, in this case, sentiment analysis. By leveraging a pre-trained BERT model, the code benefits from the model's contextual understanding of language, enhancing its ability to capture sentiment nuances in tweet texts.
7. **Model Inference and Prediction** The code saves the trained model, loads it for inference, and predicts sentiment for a sample text.

**Summary:**

* **eval\_loss:** The evaluation loss on the validation dataset is 1.718191146850586.
* **eval\_accuracy:** The accuracy of the model on the validation dataset is approximately 72.93%.
* **eval\_precision:** The precision of the model on the validation dataset is approximately 72.38%.
* **eval\_recall:** The recall of the model on the validation dataset is approximately 72.93%.

**Approach 2:**

* **FINE TUNED WITH LLM MODEL GPT3**

1. **Problem Statement and Approach:** The objective is to perform sentiment analysis on tweet texts using the GPT-3 language model. The approach involves data preprocessing, fine-tuning the model, and utilizing the fine-tuned model for sentiment analysis on new input prompts.

* Stripping extra whitespaces from tweet text and sentiment labels.
* Modifying tweet text by appending an "Intent:" section.
* Modifying sentiment labels to include a starting space and "END."

1. **Data Formatting:**

* Dropping unnecessary columns ('emotion') from the dataset.
* Renaming columns to 'prompt' and 'completion' for GPT-3 formatting.
* Saving the preprocessed data in JSONL format for training.
* Using OpenAI tools to prepare data for fine-tuning.
* Setting the OpenAI API key for accessing GPT-3.

1. **Fine-Tuning Process:** The fine-tuning process involves preparing and training the GPT-3 language model on the sentiment analysis task.

* Creating a fine-tuning job with specified training and validation data, computing classification metrics, and defining the number of classes.

4.**Model Usage:** After fine-tuning, the model is ready for sentiment analysis on new data. Setting up the prompt for sentiment analysis with an "Intent:" section. Utilizing the OpenAI API to create a completion using the fine-tuned model. Extracting and printing the sentiment analysis result.

5. **Model Output:** The output of the sentiment analysis for the given prompt, "iPhone is not working properly," is classified as "Negative emotion."

**Deploying a FastAPI application on AWS**

To deploy app in aws you can use elastic beanstalk if you don’t have any system requirements. Otherwise you can use by create EC2 instance where you need to do manual configurations.

**Step 1: Create an EC2 Instance**

* Log in to the AWS Management Console: Navigate to the EC2 Dashboard.
* Launch an Instance: Click on the "Launch Instance" button. Choose an Amazon Machine Image (AMI) based on your system.
* Choose a Key Pair: In the key pair section, either create a new key pair or choose an existing one. If creating a new key pair, download the .pem file.
* Review and Launch: Review your instance configuration and launch the instance.

**Step 2: Connect to the EC2 Instance**

* Get the Public DNS: Once the instance is running, note the Public DNS (IPv4) from the EC2 dashboard.
* SSH into the Instance: Open a terminal on your local machine. Use the following command to connect to your EC2 instance:

ssh -i /path/to/your/key.pem ec2-user@your-public-dns

**Step 3:Copy Your FastAPI App to the EC2 Instance:**

Use scp or any other method to copy your FastAPI app files to the EC2 instance. Install FastAPI and Uvicorn: Install FastAPI and Uvicorn using pip.

Your FastAPI app should now be running on the EC2 instance. Access it by opening a web browser and going to http://your-public-dns:8000/docs to view the interactive API documentation.