Analyze emotion and product

In my notebooks you can find that I have experimented with about 4 models(Naive Bayes, Random Forest, Vader and Roberta).However, to proceed with my prototype on Flask I am going to make use of VADER (Valence Aware Dictionary and sentiment Reasoner - Bag of words approach)from the nltk library.

The reason I chose VADER is because it's sensitive to both the polarity (positive/negative) and the intensity (strength) of emotions expressed in text, particularly well-suited for sentences like these where sentiments are conveyed through both individual words and their combination.

For the brand identification I have created a parser function to categorize text into ‘Apple’, ‘Google’ or ‘No Brand identified’. A better approach was to make use of a BERT model and modify it to output a label for each category in the tweet sequence, not just a single label for the entire sequence.

The notebooks (wysa.ipynb and wysa2.ipynb) can be found in my github link :
https://github.com/rohan-patnaik/sentiment-analysis-using-VADER

I will be reporting my findings now. (Detailed EDA with plots can be found in the notebooks).

- The EDA shows a high focus on tech products, particularly from Apple and Google.
- Most of the tweets in this excel show positive sentiments towards these brands.
- A big subset of these tweets contain tweets with 'No emotion towards any specific brand/product'.
- More than 70% of the tweets have mentioned the hashtag #sxsw.On further research I found that South by Southwest, abbreviated as SXSW, is basically a large organized film festival that takes place in mid-March in Austin, Texas, United States every year!

Conclusion:
Based on this we can conclude that this dataset contains tweets heavily from the people who are visiting SXSW. To further make our evidence more concrete this EDA chart below shows that the tweets sent from mobile devices(be it Apple or Google android) are more than 90% of the count.

Suggestions to Improve the Dataset:

Handling Missing Data:
To enhance the dataset, we can start with the missing values in 'emotion_in_tweet_is_directed_at'. Perhaps it's something about how the data was gathered or the tweets themselves. We could try filling in these gaps with educated guesses (imputation), but we should tread carefully because there's quite a bit missing. Another intriguing idea is to create a model that predicts these missing pieces based on the content of the tweets.
Data enhancement:
Enriching the data could really open up new perspectives. Imagine including when the tweet was posted, who posted it, and how much buzz it created (like likes and retweets). This could really spice up our analysis.
Text Preprocessing:
Further preprocessing of tweet text (like removing special characters, URLs, or standardizing text) could be beneficial, especially for text analysis or machine learning models.
Categorization and Sentiment Analysis:
Getting more nuanced with our emotion categories and doing a sentiment check on the tweet texts could give us a clearer picture of how people feel about different brands or products.
Sentiment analysis on tweet text to align with the emotional categories could validate the current emotion labels.
Data Quality Checks:

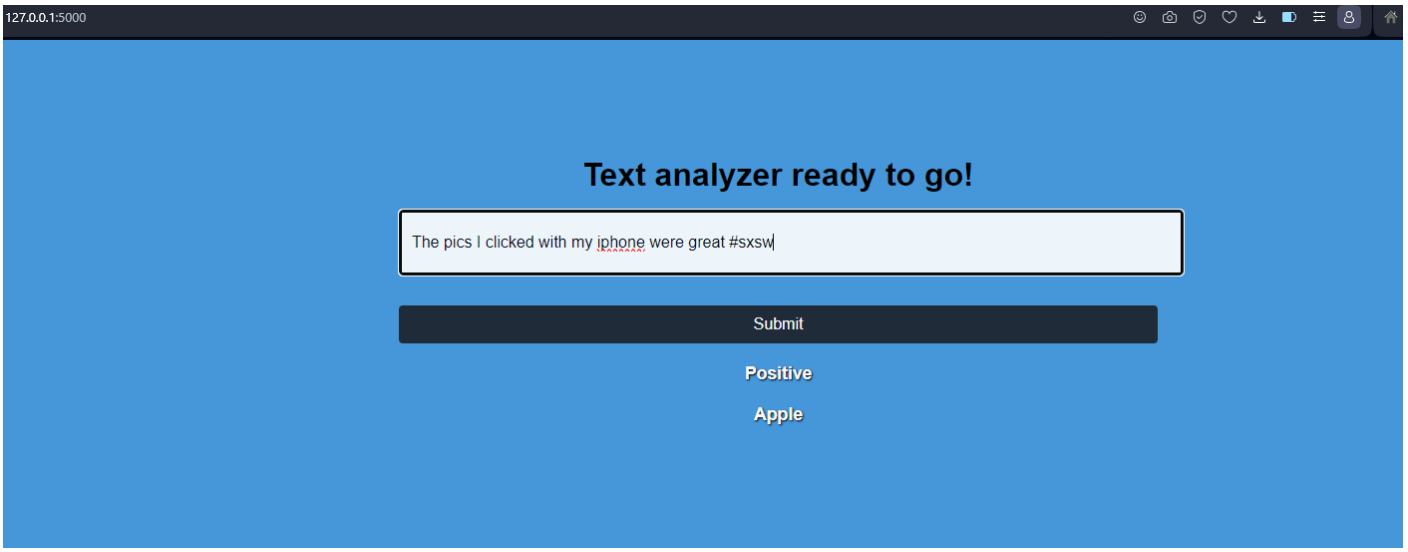
Ensuring that all tweets are relevant to the context (e.g., related to technology, brands, etc.) and filter out any irrelevant data.
Check for duplicates or near-duplicates in tweet text to ensure data quality.

Q. Explain your approach to evaluate the trained model.
Ans. To evaluate the model, we could set aside a portion of the training data as a validation set. Here are some evaluation strategies:

- Confusion Matrix: To visualize the performance of the classification model across all classes.
- Classification Report: To get precision, recall, and F1-score for each class, which are critical when dealing with imbalanced datasets.
- Accuracy Score: To get an overall idea of the model's performance but with caution due to potential class imbalance.
- Cross-Validation: To assess the model's robustness and generalizability across different subsets of the data.

Keep in mind that for sentiment analysis, especially with VADER, manual inspection of the results can also be valuable. You might randomly select some predictions and review them to see if they make sense intuitively. This qualitative analysis can reveal things that quantitative metrics might miss.

Q. Plus: if you have a working prototype by serving your model with Flask, FastAPI or something similar.
Ans. I have tested out the prototype with Flask deployment and have pushed the code to github for you to view.
For the time being I am attaching a small screenshot of a simple web application created using VADER and FLASK doing sentiment analysis ->



Q. Write about the requirements and ways to deploy the model to prod. Also explain about different types of metrics someone should monitor after deployment.

Ans. Requirements for Deployment:

1. Model Format: Ensure the model is saved in a suitable format for deployment.
2. Deployment Environment: Choose an appropriate environment for deployment, such as a cloud platform (AWS, Google Cloud, Azure)or a containerized environment (Docker).
3. API Development: Develop an API (usually RESTful) for the model using frameworks in Python. The API should handle requests with input data and return the model's predictions.

4. Scalability: Ensure the infrastructure can scale to handle varying loads.
5. Security: Implement security measures to protect the API and the data it processes. This includes authentication, encryption (HTTPS), and secure handling of sensitive data.
6. Logging and Monitoring: Set up logging and monitoring to track the API's usage, performance, and errors.
7. CI/CD Pipeline: Establish a continuous integration and continuous deployment (CI/CD) pipeline for streamlined updates and maintenance of the model and API.

Post-Deployment Monitoring Metrics:

1. Latency: Monitor the response time of the API. High latency might indicate a need for optimization or scaling.
2. Throughput: Track the number of requests handled per unit time. This helps in understanding the load and planning for scaling.
3. Error Rates: Keep an eye on API errors and failed requests. High error rates may indicate bugs or issues in the model or API.
4. Resource Utilization: Monitor CPU, memory, and disk usage to ensure the infrastructure is not over/underutilized.
5. Model Performance Metrics: Track model-specific metrics like accuracy, precision, recall, and F1 score. You might need to periodically reevaluate the model against a validation set or use live feedback.
6. User Feedback: Collect and analyze user feedback for qualitative insights into the model's performance in real-world scenarios.