Sentiment Analysis (Task 3)

MoveInSync Assessment

Vanshvardhan Singh

iMTech 5th year, CSE with specialization in AI/ML

Table of contents

Exploratory Data Analysis

EDA

- There are 1,54,278 inputs in training data. Out of these 17,397 inputs in training data are null and 8696 inputs in training data are duplicate.
- A lot of our data is in Hinglish. Which makes using pre-trained word embeddings harder.

Table 1: Categories and their Value Counts in traning data

Category	Value Count
Positive	68,228
Neutral	52,318
Negative	33,732

Observation

There is a skew towards positive and neutral classes.

Word Cloud

(a) Positive sentiment

(b) Neutral Sentiment

(c) Negative Sentiment

Figure 1: Wordclouds

Word Length Histogram

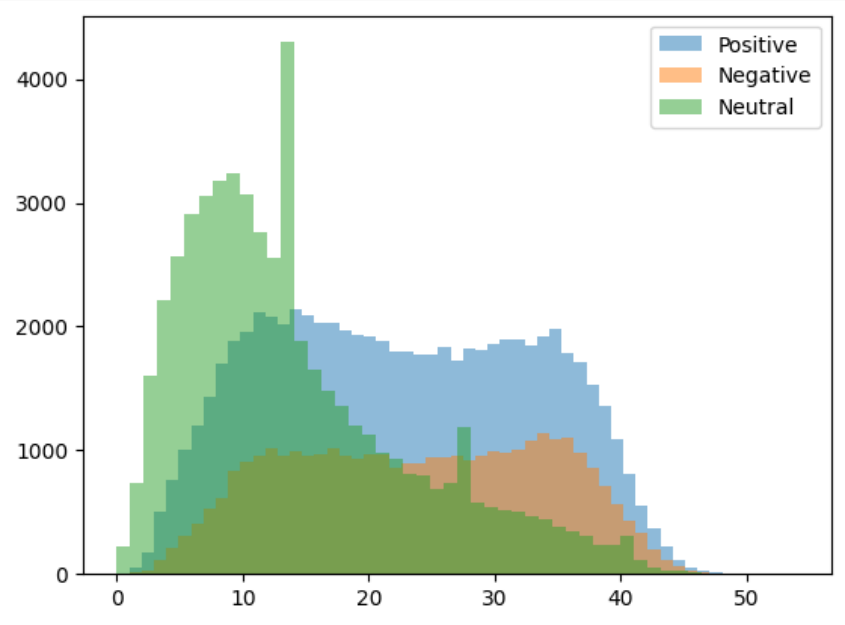


Figure 2: A Histogram showing the words counts of texts, divided by sentiments

Observation

Word count might be a good feature to add as part of feature engineering.

Preprocessing

Preprocessing

- NaN Value Removal
- URL removal
- Expanding contractions
- Lemmatization
- Removing punctuations
- Removing Duplicate texts
- Adding Word Length as a feature
- SMOTE oversampling (disregarded, gave poor results)

Vectorization

Vectorization

Some ways we could convert text into numbers which our model could work with are

- **TF-IDF**: Short for Term Frequency-Inverse Document Frequency. It measures a term's importance by how often it appears in a document versus across the corpus. A key drawback is that it ignores the word order.
- **Word Embeddings**: We can use models like Word2Vec or GloVe for static embeddings or transformer-based models like GPT for contextual embeddings that change meaning based on context.

Problem Description

These are the two ways we can frame the problem, along with their advantages.

- **Regression** We do this to exploit the fact that neutral sentiment is somewhere 'in between' positive and negative sentiment.
- **Multi-class classification** This handles multiple labels naturally. It gives a probability distribution of confidence intervals. Furthermore, it avoids threshold as an additional hyper-parameter.

Training

Regression Models

Table 2: Regression models results

Model	Features	Accuracy
Linear Regression	10,000	71.8%
Neural Network	500	55%

Observations

- Regression models are slow to train and thus slow to tune for hyper parameters.
- Neural Network's performance suffers because we are not able to pass a sparse matrix for training.

Classification Models

Table 3: Classification models results

Model	Features	Accuracy
Ridge Classifier	10,000	87.8%
Naive Bayes	3,000	69.3%
XGBoostClassifier	10,000	88.1%

Observations

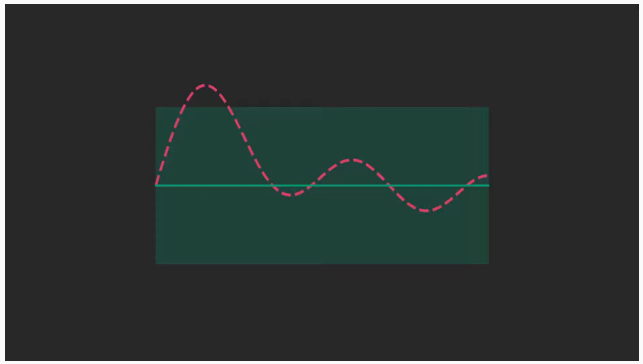
- Classification models perform much better than regression models.
- Ridge Classifier takes advantage of the fact that the neutral point lies between negative and positive (what we intended to do).
- XGBoost performs reasonably well. We can select this model and move on to parameter tuning.

Parameter Tuning

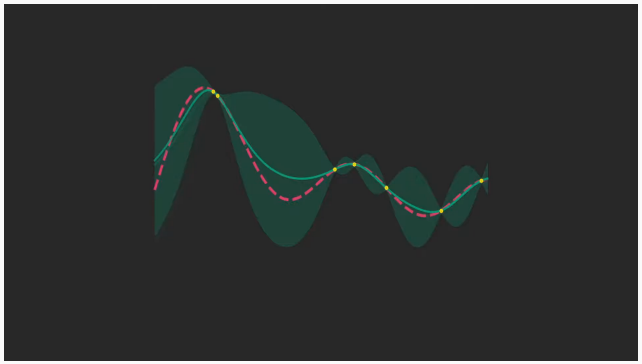
GridSearchCV is an option, but this much more time to compute. We are going to use Bayesian Optimisation for parameter tuning.

Bayesian Optimization builds a probabilistic model (Gaussian Process) to estimate the behavior of a function.

Bayes Optimisation



(a) Initial state space of loss function



(b) State space after a few iterations

The pink line is the true function (actual loss).

The green area represents possible values the pink line might take, reflecting uncertainty.

It selects the next evaluation point by balancing:

Exploration: Testing uncertain areas (wide green regions).
Exploitation: Focusing on areas likely to improve results (low predicted values).

Redefining Score

Now that we are approaching 90% accuracy, we can use other metrics instead of accuracy for measuring our model. From now on, we are going to use ROC-AUC for hyper parametre tuning.

ROC Curve: Plots the trade-off between True Positive Rate and False Positive Rate as the decision threshold changes.

AUC (Area Under Curve): A single number summarizing the curve; higher values mean better separation.

Why use it?

It's threshold-independent, evaluating performance across all possible thresholds. Focuses on ranking predictions, making it useful when class distributions are imbalanced.

Results

While computing the final results, we increase the max features value in our TF-IDF vectorizer to 30,000. I could run 100 iterations of bayes optimisation before the Google Colab runtime disconnected.

Table 4: Best Performing model Results

Metric	Value
Accuracy	0.9291
Precision (Weighted)	0.9287
Recall (Weighted)	0.9291
F1 Score (Weighted)	0.9287
ROC-AUC (OVR)	0.9830

Possible Improvements

Possible Improvements

- Could remove Hinglish text and convert it to pure English. We don't need a transformer for this since the text length doesn't increase significantly, but still, we may need an RNN or LSTM architecture.
- Use word embeddings (Word2Vec, GloVe). Currently, our dataset contains Hinglish words, so we can't use pre-trained embeddings.
- Use state-of-the-art Hugging Face models like DeBERTa.
- Wherever a dense matrix is required, we could apply PCA to reduce dimensions and then convert it to a dense matrix. This would result in a more compact representation of information.