ISyE 6740 – Spring 2024
Project Report

Team Member Names: Mahshid Jafar Pour, Mehdi Sadeghi (Project Team 159)

Project Title: **Extracting Emotions from Twitter Posts: A Multi-class Classification NLP Problem**

Problem Statement:

Sentiment analysis is a classic NLP (Natural Language Processing) problem where the goal is to find the sentiment expressed in the text. Often, it is run on social media posts, product reviews or comment sections of a news article. This is crucial for businesses to understand public opinion on a particular topic or product. Companies often use Sentiment analysis to track customer satisfactions, monitor social media, perform market research or analyze employees survey [1].

Although sentiment analysis is widely used, it has its own challenges: detecting sarcasm in text is hard; emojis are used in text more than ever which cause difficulties in accurate classification; language biases; dealing with large volume of data; use of words that have multiple meanings which can be deceiving to the model; negation does not always mean a negative sentiment, etc. [2]. The trick is to train the model on a large enough dataset and perform the pre-processing stages cautiously.

In this project, we are aiming to run a similar problem by focusing on the content shared on X (former Twitter). However, instead of positive, negative and neutral labeling for the text, we are assigning emotions to them such as love, hate, anger, etc. In other words, our approach involves utilizing various multi-class classification algorithms to precisely categorize Twitter posts based on their emotional content.

Data Source

We have used "Emotions" dataset, which is shared on Kaggle [3] for our project. Contrary to traditional positive/negative sentiment analysis, this dataset has six distinct emotion categories: sadness (0), joy (1), love (2), anger (3), fear (4), and surprise (5). It serves as a valuable resource for understanding and analyzing the diverse spectrum of emotions expressed in short-form text on social media.

This dataset has 416,810 messages shared on Twitter, each entry consists of a text feature accompanied by a corresponding emotion label. It is important to note that the dataset is unbalanced, with a greater percentage of texts expressing sadness and joy compared to other emotions. This imbalance requires careful consideration during model development and evaluation to ensure robust performance across all emotion categories.

Methodology

Python is the main language used for this project. The steps involved will include:

1) Data Pre-processing

As the initial step, we performed data pre-processing to clean the data, and ensure optimal data quality and consistency. This involves a series of steps including converting all text to lowercase to eliminate case sensitivity and ensure uniformity. Additionally, whitespace removal is necessary to eliminate unnecessary whitespace characters to improve readability, removal of hashtags, "RT" and hyperlinks, removal of punctuations, replacing chatword abbreviations with their full words (e.g. "you" instead of "u"), removal of English stopwords ("is", "a", "the", "are", etc.), misspelling correction and finally lemmatization was applied.

During EDA stage we noticed that apostrophe character was removed from the original dataset; therefore, more stopwords (such as "dont") were added to the standard ones to include stowords without aphostrophe. Spell checking was the most computationally-intensive step in the pre-processing. In addition to automatic spell checking, about 400 words, that were not recognized by SpellChecker library were manually corrected, these words include intentional misspelling such as "realllly" instead of "really".

2) EDA

Once the data are cleaned and pre-processed, we can perform Explanatory Data Analysis (EDA) to have a better understanding of data distribution.

An initial bar-plot of labels show that data are imbalanced. Imbalanced data creates the challenge of learning the small classes (surprise and love) by model correctly.

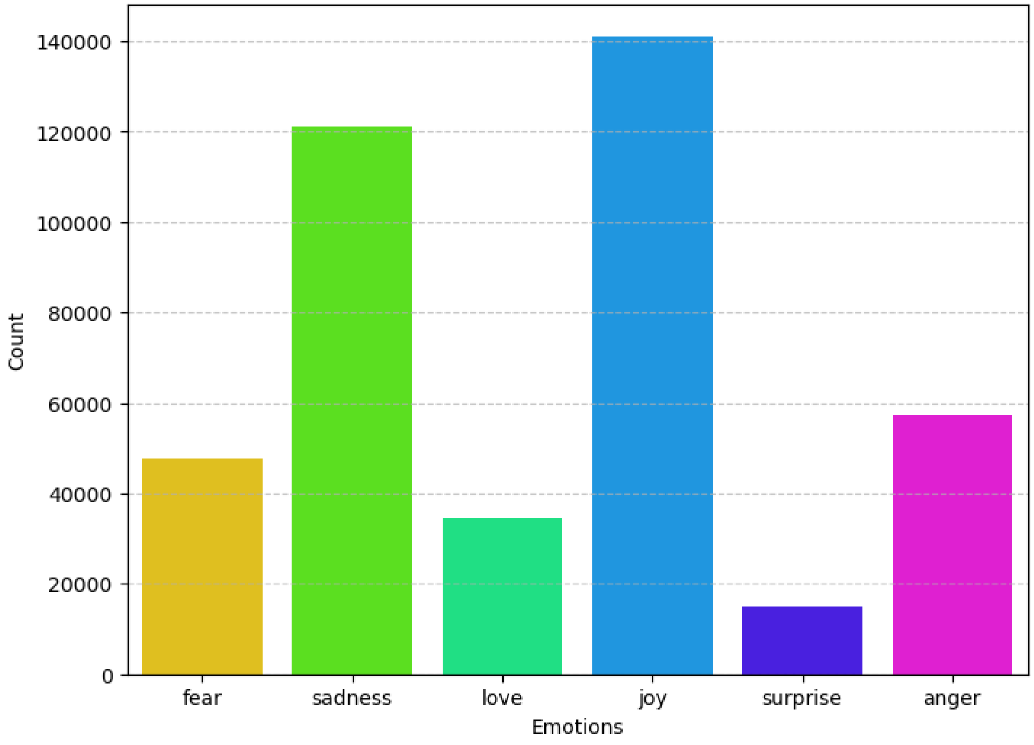

Figure 1 Imbalanced data (Surprise and Love are the least represented emotions in the dataset)

Word cloud for each category is also generated, these visualizations show the most frequently-used words in each category. One sample is shown below:

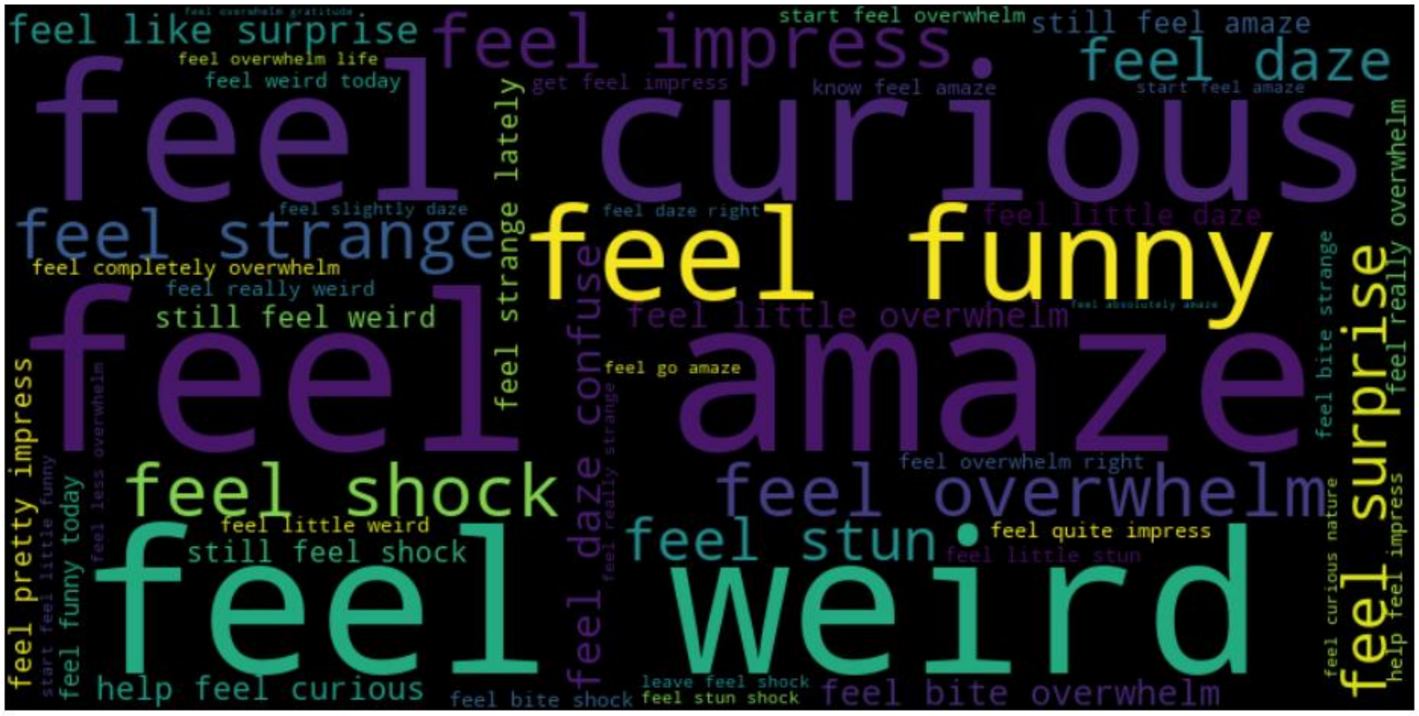

Figure 2 Word cloud for "surprise" category.

As our data is unbalanced, we have experimented with some undersampling and oversampling techniques to overcome this challenge and ensure the final model learns effectively and improve its performance on classifying messages in correct emotions.

3) Feature Extraction

The next step is feature extraction, we have explored different methods of word embedding to convert words into numerical vectors as explained below:

3-1 Word2Vec & Doc2Vec

In word2vec embedding, the entire corpus is scanned, and the vector creation process is performed by determining which words the target word occurs with more often. It reveals the semantic closeness of two words to each other. It is based on a CBOW (Continuous Bag of Words) where it learns to predict a target based on the context. Word2vec is trained using a shallow neural network with a single hidden layer. During training, the model adjusts the word vectors to minimize the loss function, which measures the difference between the predicted and actual words in the training data. Once trained, the word vectors capture semantic relationships between words, such as synonyms, antonyms, and analogies [4].

Doc2Vec is an extension of Word2vec. It learns fixed-length vector representations for variable-length pieces of text, such as sentences, paragraphs, or documents. The idea behind Doc2Vec or document embedding models is to capture the semantic meaning and context of entire documents in a lower-dimensional vector space. We have used DBOW (Distributed Bag of Words) with a vector representation of size 300. It was trained based on 30 epochs and a variable learning rate reduced at each epoch.

3-2 TF-IDF

TF (Term Frequency) is the ratio of the term to the total number of terms in each document. IDF (Inverse Document Frequency) is the logarithm of the ratio of the total number of documents to the number of documents in which the target term occurs. Tf-idf in sklearn package is defined as:

$$tf - idf(t,d) = tf(t,d) \times idf(t)$$

$$tf(t,d) = \frac{\#\ of\ times\ term\ t\ appreard\ in\ document\ d}{total\ \#\ of\ items\ in\ document\ d}$$

$$idf(t) = log\left(\frac{n+1}{df(t)+1}\right) + 1$$

where *df(t)* is the document frequency of *t*, and *n* is the total number of documents.

Contrary to Bag of Words methods that treats all words equally, TF-IDF takes the importance of a word into consideration, not just the occurrence of a word in a single document but in the entire corpus. However, TF-IDF does not capture the semantic of the word or text context [4].

3-3 MPNET (Multi-Prospective Network)

MPNET is dense vector representation of sentences and paragraphs pre-trained on large corpora using unsupervised learning. It takes into account the context and surrounding words for embedding. MPNET architecture is based on Transformers with multi-perspective self-attention mechanisms. Each sentence in the text is represented by a dense vectors of size 768.

3-4 Keras text_to_sequences

This method first tokenizes the text into tokens and then it is fit to the entire text to map each unique word to an integer index.

4) Model Training

Embedded data are partitioned into train (80%), and test (20%) datasets for model training, model selection, and accuracy measurements. Since data are imbalanced, we divided data with stratified sampling to ensure that the distribution of classes in both sets is representative of the original dataset.

We have used different classical machine learning classifiers (such as Multinomial Naïve Bayes, support vector machine (svm), svm with rbf kernel, logistic regression, knn with cosine similarity and random forest) as well as deep learning for model training. For deep learning, we used a 4-layer neural network (NN) with one embedding, one LSTM (Long Short-Term Memory) and feedforward layers. Batch normalization and drop-out were also used in the model architecture. We chose Adam optimizer.

Additionally, we explored open-source pre-trained Large Language Models (LLMs) for the purpose of transfer learning and potentially achieving a better result leveraging LLMs to classify our data. In particular, we have used pre-trained LLaMA (Large Language Model Attack) for sequence classification [5] with MPNet text embedding.

Cloud was used for model training and all neural networks were trained on GPU because of the size of data and dense vectorization of texts.

Evaluation and Final Results

Different metrics are used to measure the performance of the models and select the best one. We chose accuracy and per-class accuracy as the main metric. However, we looked at confusion matrix, precision, recall and F1 score to select the best model. Since we have class imbalance, it is important to get a better class accuracy for the smallest class, rather than overall accuracy. Therefore, we are reporting accuracy of the two smallest classes (Surprise and Love) as well. Table 1 summarizes the metrics on test dataset for all the models trained on the entire imbalanced train dataset.

Table 1 Models performances summary

Feature Engineering	Model	Overall Accuracy	Precision	Recall	F1-Score	Surprise Accuracy	Love Accuracy
Doc2Vec	Logistic Regression	0.6445	0.64	0.64	0.64	0.4408	0.3355
Doc2Vec	Linear SVM	0.5848	0.57	0.58	0.54	0.0692	0.0567
Doc2Vec	KNN[1]	0.6989	0.70	0.70	0.69	0.3633	0.4843
Doc2Vec	Random Forest[2]	0.5006	0.64	0.50	0.41	0.0000	0.0000
TF-IDF	Multinomial Naïve Bayes	0.7775	0.81	0.78	0.75	0.0932	0.2651
TF-IDF	Logistic Regression	0.8864	0.89	0.89	0.89	0.6789	0.7333
TF-IDF	Linear SVM	0.8579	0.87	0.86	0.85	0.5851	0.4523
TF-IDF	Kernel SVM[3]	0.8613	0.86	0.86	0.86	0.6845	0.6600
TF-IDF	KNN (n=5)[1]	0.7492	0.75	0.75	0.74	0.4248	0.4788
TF-IDF	Random Forest[2]	0.8514	0.85	0.85	0.85	0.6481	0.6203
MPNET	LLaMA	0.7560	0.76	0.76	0.76	0.6233	0.4989
Tokenizer[4] with text to sequence	LSTM NN	0.9295	0.94	0.93	0.93	0.9043	0.7139

[1] Cosine similarity is used for knn with n=5.

[2] 100 estimators are used for Random Forest.

[3] rbf kernel is used.

[4] 80,000 words used for tokenizer.

LSTM NN gave the highest metrics with an accuracy of ~93%, followed by logistic regression with an accuracy of ~89%. Figure 3 shows the confusion matrix for the best model. Figure 4 demonstrates model training curves, accuracy and loss with each epoch.

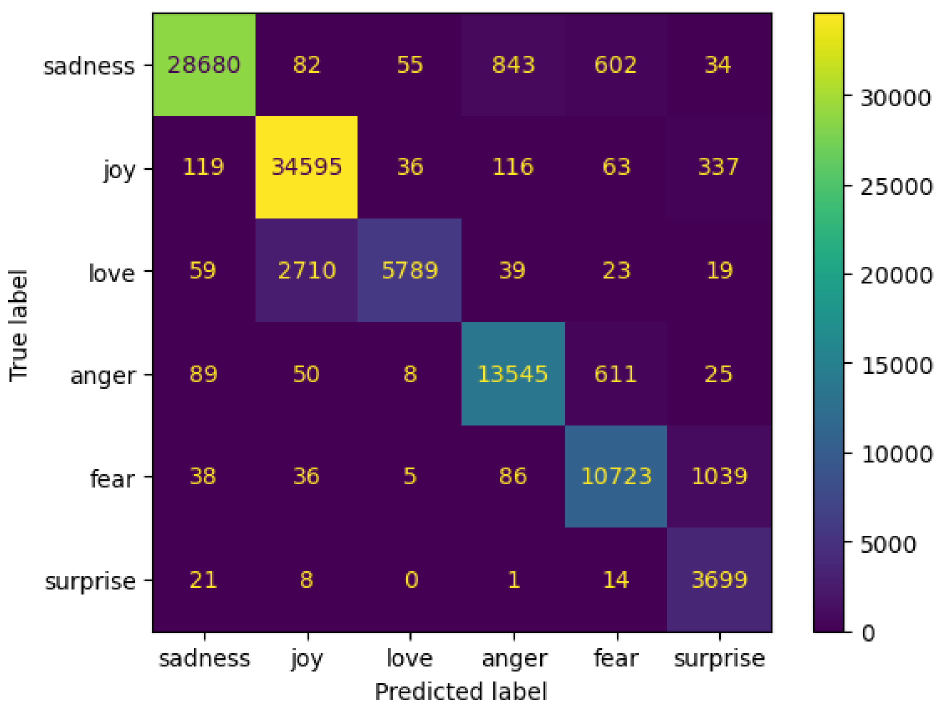

Figure 3 Confusion matrix for LSTM NN model.

From the results, we can tell that the appropriate embedding technique, tailored to the data and application, has a huge impact on the overall accuracy. We can see that all the models performed better with TF-IDF embedding compared to Doc2Vec.

It is surprising to see that LLaMA only achieved an accuracy of 76%, but it is noteworthy that it has a pretty high (0.62) minority class accuracy. It could be that the NN architecture requires more fine-tuning.

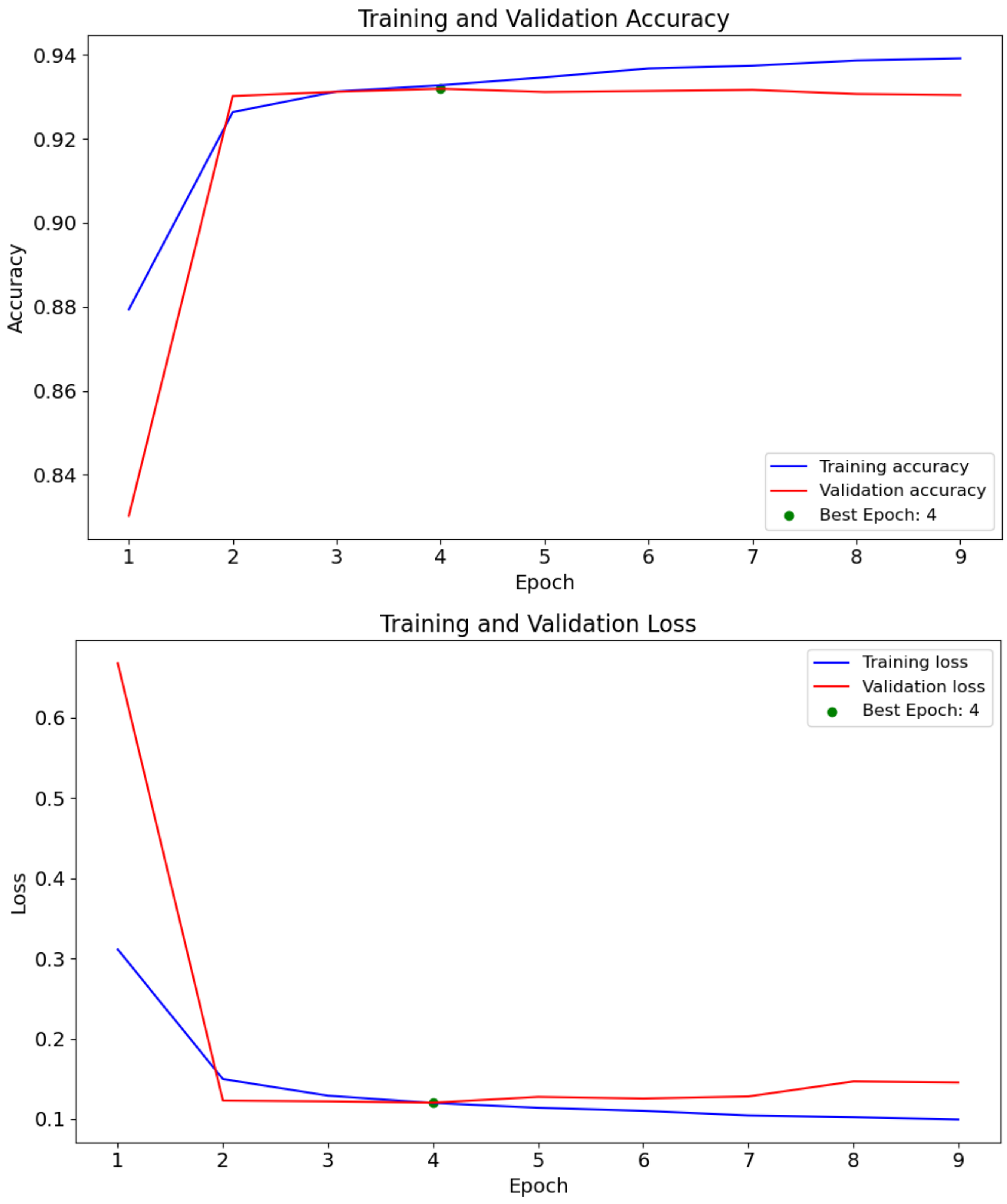

Figure 4 Accuracy and loss with epoch for training and validation datasets (LSTM NN model).

Except for kernel SVM, training classical ML models were fast and straightforward. On the contrary, training deep learning models were computationally intensive, mainly because of the size of large vectors created by MPNET and Keras text_to_sequence embedding. Overall, if we want a quick sentiment analysis, it may be best to use TF-IDF with a linear classifier. But a properly tuned deep learning model with an appropriate dense vectorization/word embedding is necessary to achieve a desired high accuracy.

Random Forest performance with doc2Vec is significantly poor compared to some simple linear models, this could be because random forests may struggle with the high-dimensional, sparse nature of text data and feature engineering techniques and the doc2vec word embeddings might not be used optimally. Random Forest is specifically sensitive to class imbalance and performs poorly on minority classes. It is also sensitive to noise in the data. As a result, filtering out rare words may help improve its performance.

Dataset Balancing by Over-sampling

The main challenge of this dataset is imbalanced nature of it, where certain classes are underrepresented compared to others. To address this issue, we investigated oversampling techniques to potentially enhance model performance. We experimented with various oversampling methods, such as Adaptive Synthetic Sampling (ADASYN), random sampling, and Synthetic Minority Oversampling Technique (SMOTE). SMOTE simply duplicates the examples of minority class [6]. Our experiments revealed the following outcomes (summarized in Table 2): Random sampling marginally enhanced the overall accuracy of the LSTM NN model, primarily by significantly improving accuracies in minority classes. Meanwhile, applying SMOTE resulted in approximately a 2% increase in accuracy for the LLaMA model. It is important to carefully consider the impact of oversampling on enhancing the model's predictive capabilities, especially for achieving improvements in accurate predictions for minority classes and ensuring consistent accuracies across all classes. No improvements were seen for ADASYN; therefore, it is not included in Table2.

Table 2 Effect of oversampling on model performance

Model	Balancing Technique	Accuracy	Accuracy Improvement	Precision	Recall	F1-score
LSTM NN	Smote Oversampling	0.9136	No	0.92	0.91	0.92
LSTM NN	Random Oversampling	0.9308	Yes 0.0013	0.94	0.93	0.93
LLaMA	Smote Oversampling	0.7748	Yes 0.019	0.77	0.77	0.77

References

1. 8 applications of sentiment analysis, https://monkeylearn.com/blog/sentiment-analysis-applications/#:~:text=Some%20popular%20sentiment%20analysis%20applications,text%20by%20emotion%20and%20opinion.
2. Begüm Yılmaz, Top 5 Sentiment Analysis Challenges in 2024, https://research.aimultiple.com/sentiment-analysis-challenges/.
3. https://www.kaggle.com/datasets/nelgiriyewithana/emotions/data

4. A guide on word embedding in NLP, https://www.turing.com/kb/guide-on-word-embeddings-in-nlp.
5. https://huggingface.co/docs/transformers/en/model_doc/llama
6. https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/