Final Project

Anna Harner, Tiffany Le, Ellie Nguyen, and Yanelly Mego

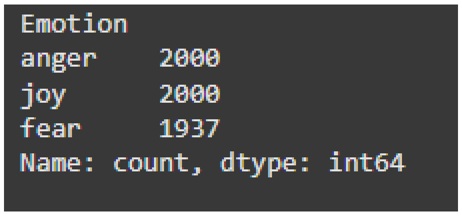
Introduction

https://www.kaggle.com/datasets/abdallahwagih/emotion-dataset

Determining the underlying emotion from text can be difficult, resulting in miscommunication and misunderstandings. That is why our project focuses on the classification of text samples into categories of emotion, focusing on anger, joy, and fear. The dataset contains a diverse collection of text samples, each labeled with an emotion it conveys. By creating a model that can interpret emotions from text, we can accomplish tasks involving sentiment analysis, emotion classification, customer feedback analysis, social media sentiment monitoring, and other emotion-aware applications.

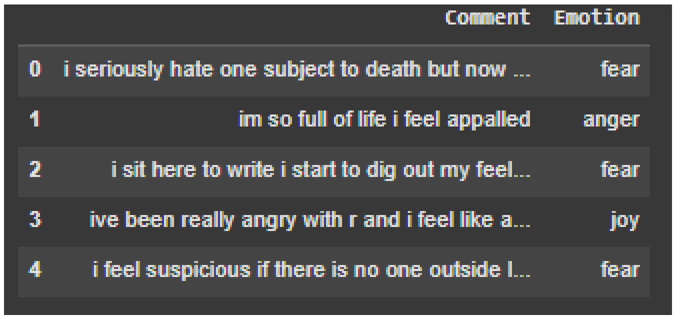
Analysis

The dataset consists of 5934 unique text samples with labels: [‘anger,’ ‘joy,’ ‘fear’]. 34% of the dataset is classified as anger, 34% with joy, and 33% with fear.



```
Emotion
anger    2000
joy      2000
fear     1937
Name: count, dtype: int64
```

It contains two columns, one for the comment and the other for the emotion.



	Comment	Emotion
0	i seriously hate one subject to death but now ...	fear
1	im so full of life i feel appalled	anger
2	i sit here to write i start to dig out my feel...	fear
3	ive been really angry with r and i feel like a...	joy
4	i feel suspicious if there is no one outside l...	fear

In order to process text data so that a machine can understand it, we need a special parsing function to clean it. This includes taking away confusing punctuation, filtering out non-alphabetic characters, and converting all letters to lowercase.

Because the model cannot understand words from the comments, we need to encode them into integers or create word embeddings that capture their semantic meaning with numbers. The TfidVectorizer function is used to convert text into feature vectors that can be fed into a model.

Methods

We build two types of models to predict emotions from text inputs and then compare their performances, what they're good at, and what they struggle with.

SVM Model Structure

SVMs (support vector machines) are models used for classification tasks. The goal of an SVM model is to separate two groups of data with a best fit hyperplane in a high-dimensional space. For this project, we want to build this model to create a hyperplane that will divide classes of text data by emotion.

MMCs (maximal margin classifiers) are a specific application of SVMs that find the hyperplane by maximizing the margin between two classes. Although this helps to separate the classes and increases the chances of classifying data accurately, the data may not be perfectly separable in a real-life scenario. The **SVC** (support vector classifier) is a more robust model that can handle data that are not always linearly separable. It does this by using kernel functions, which include linear, polynomial, and RBF. SVCs also tolerate some level of error with **slack variables,** which allows data points to lie within or on the wrong sides of the margin.

The hyperparameters of an SVC model include:

1. `C`
2. `gamma`
3. `kernel`

In the SVC model, a **kernel**'s type is specified by the `kernel` hyperparameter**.** The **kernel** projects the data into a higher dimensional space in which they are separable by a hyperplane.

Given the coefficient and dimension, it transforms the data by squaring every value, calculating the relationship between variables, and then projecting that to a higher dimension.

The `C` hyperparameter acts as a regularization value that determines how much error is tolerated in training, while the `gamma` hyperparameter scales how much influence we want the data points to have on each other.

In order to choose the best hyperparameter values for our SVC model, we used the **GridSearch** method. This method decides hyperparameter values by calculating the accuracy for every possible combination of hyperparameter values, which we specify in a parameter grid, such as the one shown below. Then, it finds the combination of values that provide the best result.

```
# Define the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100],          # Regularization parameter
    'kernel': ['linear', 'rbf'],    # Kernel types
    'gamma': ['scale', 'auto'],     # Kernel coefficient (relevant for
'rbf')
}
```

We configured our GridSearch to optimize for accuracy and used 5-fold cross-validation to split the data into multiple folds before training and validating the data on these different subsets. This is a reliable but computationally expensive method. The hyperparameter values chosen by GridSearch are shown below, with the best cross-validation score of 0.933.

1. `‘C’ = 1`
2. `‘gamma’ = ‘scale’`
3. `‘kernel’ = ‘linear’`

We use these hyperparameters to create an SVC model that takes text data and classifies it.

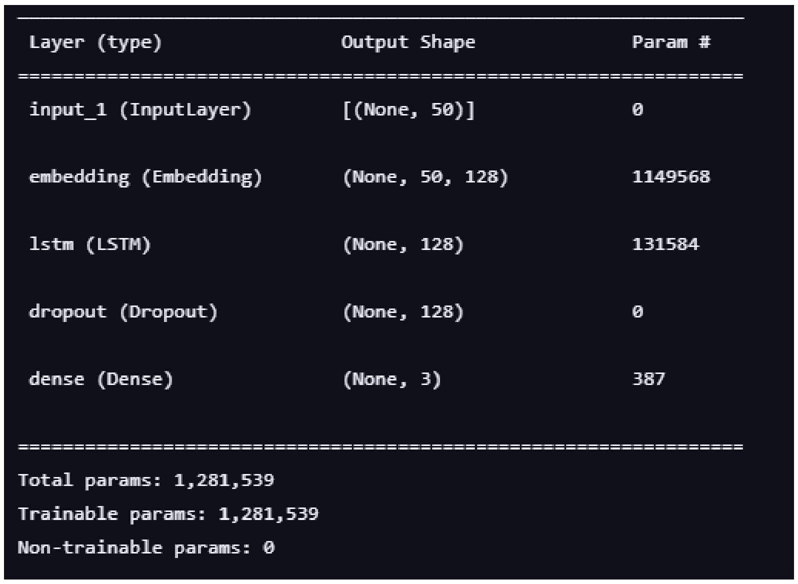
LSTM Model Structure

The next model we build is typically used in classification tasks involving sequential data. In the case of predicting the next items of a sequence, for example, in weather forecasting or story writing, the LSTM model makes a classification decision each time it generates a new item in a sequence. Based on the probability distribution, it predicts one of many possible items to produce next, making it a multi-class model. In our case, we want it to take text data and predict one of 3 emotions.

Before feeding them into the model, the texts are tokenized and converted into sequences. Then the labels are encoded. This allows us to convert textual data into numerical representations in order for the model to process them.

Additionally, class weights are calculated to help the model avoid class imbalances and learn better representations for underrepresented data.

The model contains an embedding layer, which learns the dense vector representations of the words in the text data. Additionally, a dropout layer is used to regularize the model and help reduce overfitting to the training data. The model summary is shown below:



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 50)]              0

 embedding (Embedding)       (None, 50, 128)           1149568

 lstm (LSTM)                 (None, 128)               131584

 dropout (Dropout)           (None, 128)               0

 dense (Dense)               (None, 3)                 387

=================================================================
Total params: 1,281,539
Trainable params: 1,281,539
Non-trainable params: 0
```

The model is trained for 30 epochs with a batch size of 32. It uses a learning rate scheduler, which reduces the learning rate when the performance plateaus to allow for a more smooth convergence.

Results

Both models performed very well, generating validation accuracies of over 80%. We use precision, recall, f1-score, and support metrics to evaluate the model's performance.

Both **precision** and **recall** scores tell us how often the model makes correct predictions within each category of emotion (anger, fear, and joy). They each have their own focus:

1. **Precision**: Out of all cases the model predicted as positive, how many were actually positive?
2. **Recall**: Out of all actual positive cases, how many did the model correctly identify?

The **f-1 score** is a combination of precision and recall, resulting in a value between 0 and 1 that measures predictive performance.

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP} + \mathrm{FN})}$$

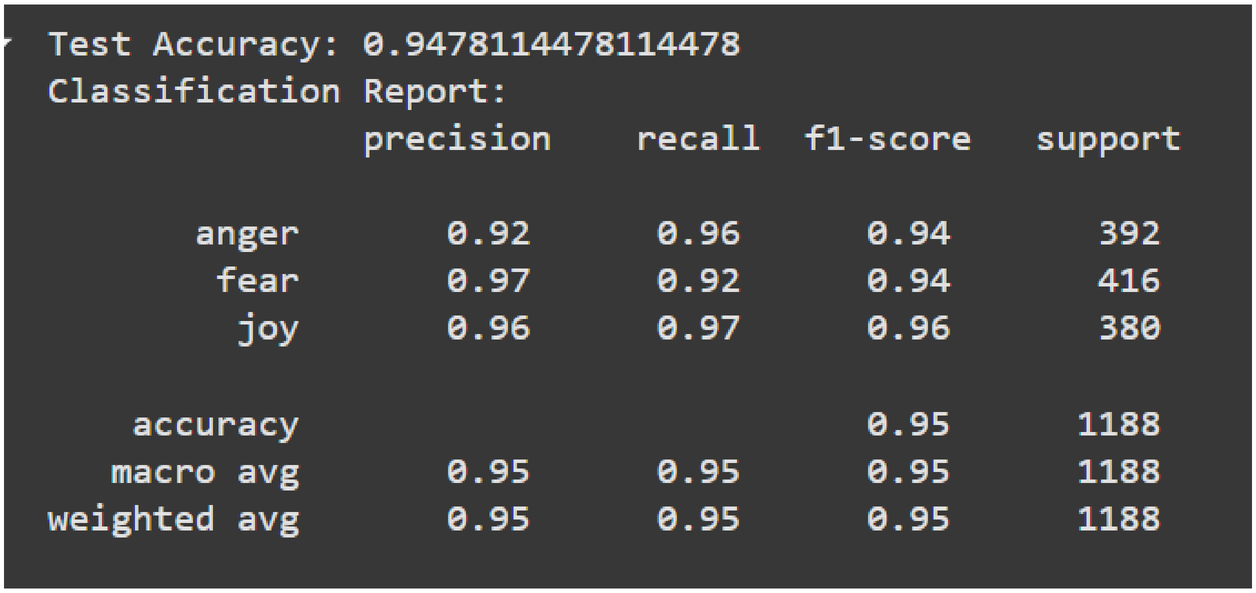
TP = number of true positives
FP = number of false positives
FN = number of false negatives

Lastly, the **support** metric refers to the number of true occurrences of each class in the dataset. It counts the number of instances for each class and is used to evaluate the performance of the model, particularly when it comes to imbalanced datasets. By including the support metric alongside the other scores, we can see how reliable the precision, recall, and f1-scores are and understand how representative the metrics are for each class.

These results are shown below:

SVM results

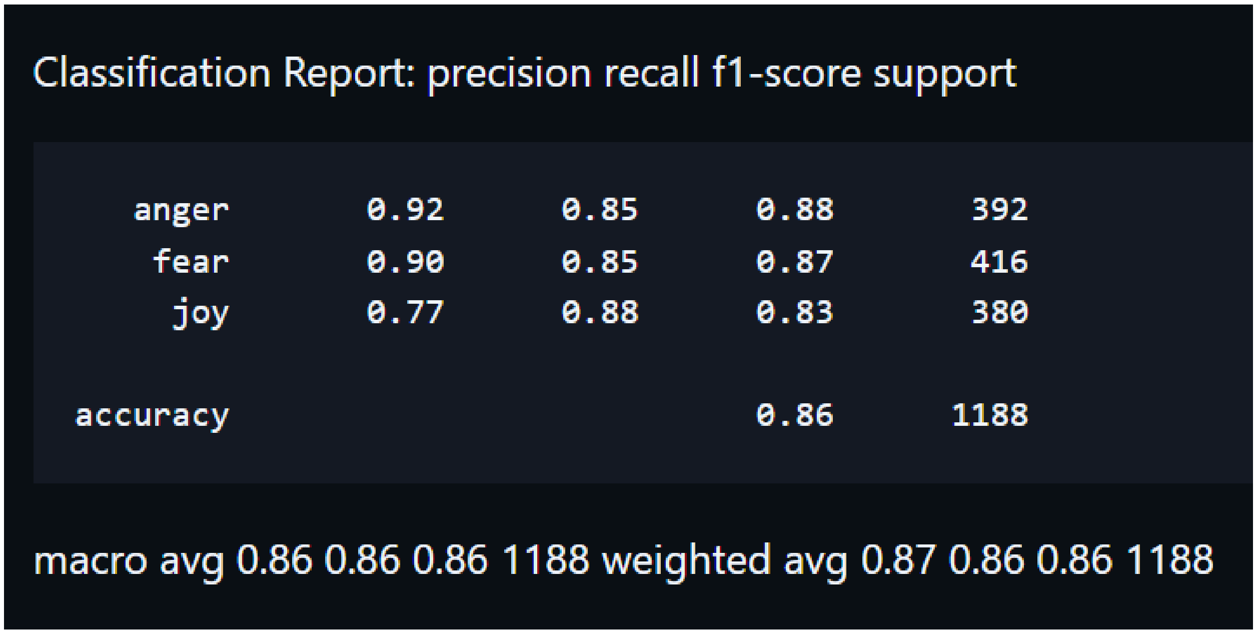
The SVM predicts fear with the best precision out of all emotions, with a score of 0.97. Its f1-score is the highest for the joy category. Overall, it seems to have an even performance score across all three categories, with a training accuracy of 95%.



```
Test Accuracy: 0.9478114478114478
Classification Report:
              precision    recall  f1-score   support

       anger       0.92      0.96      0.94       392
        fear       0.97      0.92      0.94       416
         joy       0.96      0.97      0.96       380

    accuracy                           0.95      1188
   macro avg       0.95      0.95      0.95      1188
weighted avg       0.95      0.95      0.95      1188
```

LSTM results

Although the LSTM model performed slightly more poorly than the SVM model, it still produced very high scores. We believe the difference in performance is because SVMs tend to perform better on smaller datasets, and our dataset only contains 3 emotion labels. The LSTM model produces better precision and f1-scores on anger and fear than on joy. However, its recall score is higher for joy than for the other two emotions. Its training accuracy is 86%.



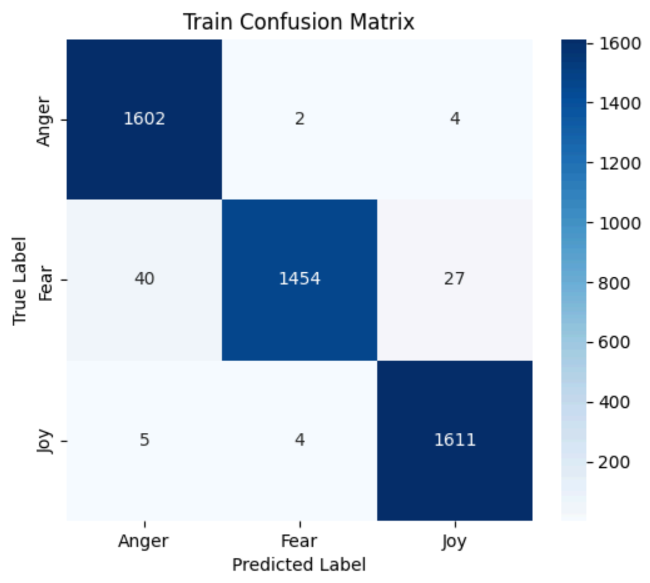
Classification Report: precision recall f1-score support

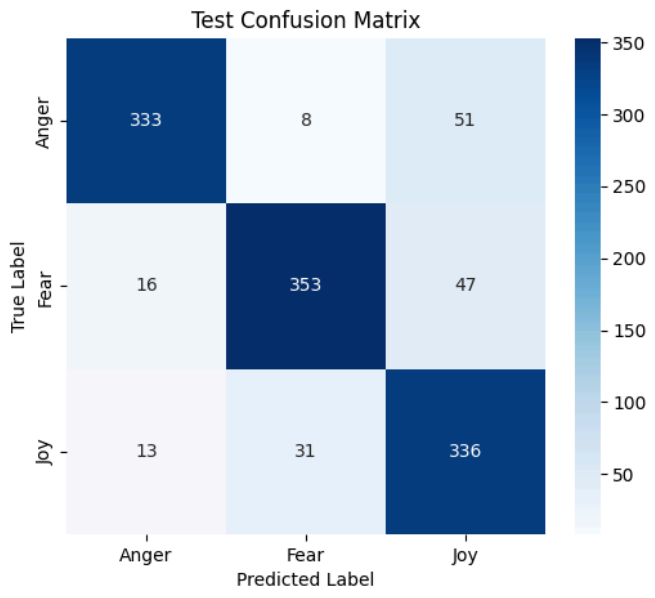
```
       anger       0.92      0.85      0.88       392
        fear       0.90      0.85      0.87       416
         joy       0.77      0.88      0.83       380

    accuracy                           0.86      1188
```

macro avg 0.86 0.86 0.86 1188 weighted avg 0.87 0.86 0.86 1188

The LSTM's confusion matrices for both train and test data also illustrate its good performance, with a high concentration of predicted labels being true across all three emotions.

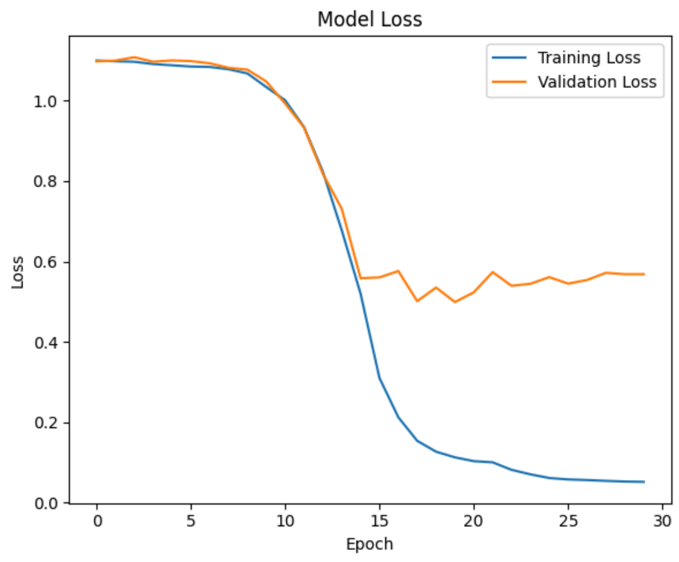
LSTM Confusion matrices:

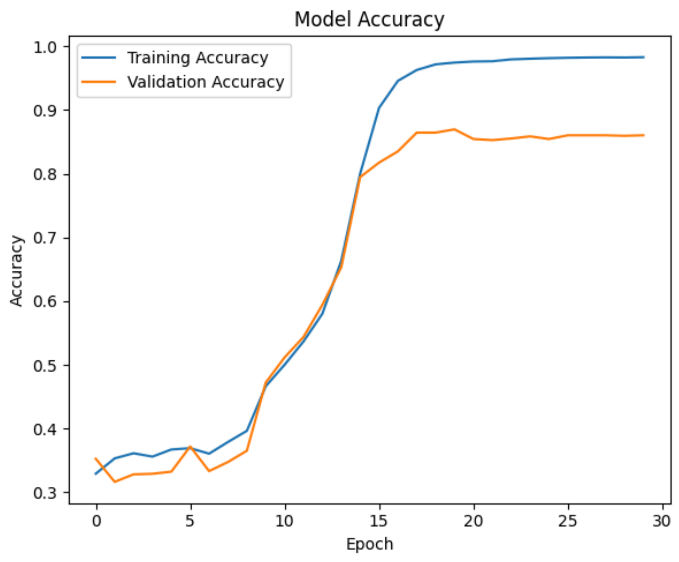




We also plotted the LSTM model's performance as it trains over epochs. As the model trains, the common trend is that the loss will decrease and the accuracy will increase. Then, both will plateau when the model converges. The graphs indicate some overfitting since the validation performance is poorer than the training performance.

LSTM accuracy and loss over epochs:





Reflection

This project allowed us to explore and compare two distinct approaches to classify emotion from text: SVM and LSTM models. Through this process, we gained valuable insight into both models, their strengths, and the challenges of each method. The SVM model demonstrated its reliability for datasets that can be represented and divided in a feature vector space; we see that this includes classifying textual data, like comments, into different emotions. On the other hand, the LSTM model demonstrated its ability to capture the sequential nature of the text, leverage word embeddings, and address class imbalances with weights. This makes it a good option for tasks where the temporal or contextual order of words significantly influences the meaning and classification. By refining these models and extending the dataset, we can enhance their effectiveness in emotional aware applications.