Comprehensive Model Report

Steven Evans

SRE220000

BUAN 6342: Applied Natural Language Processing

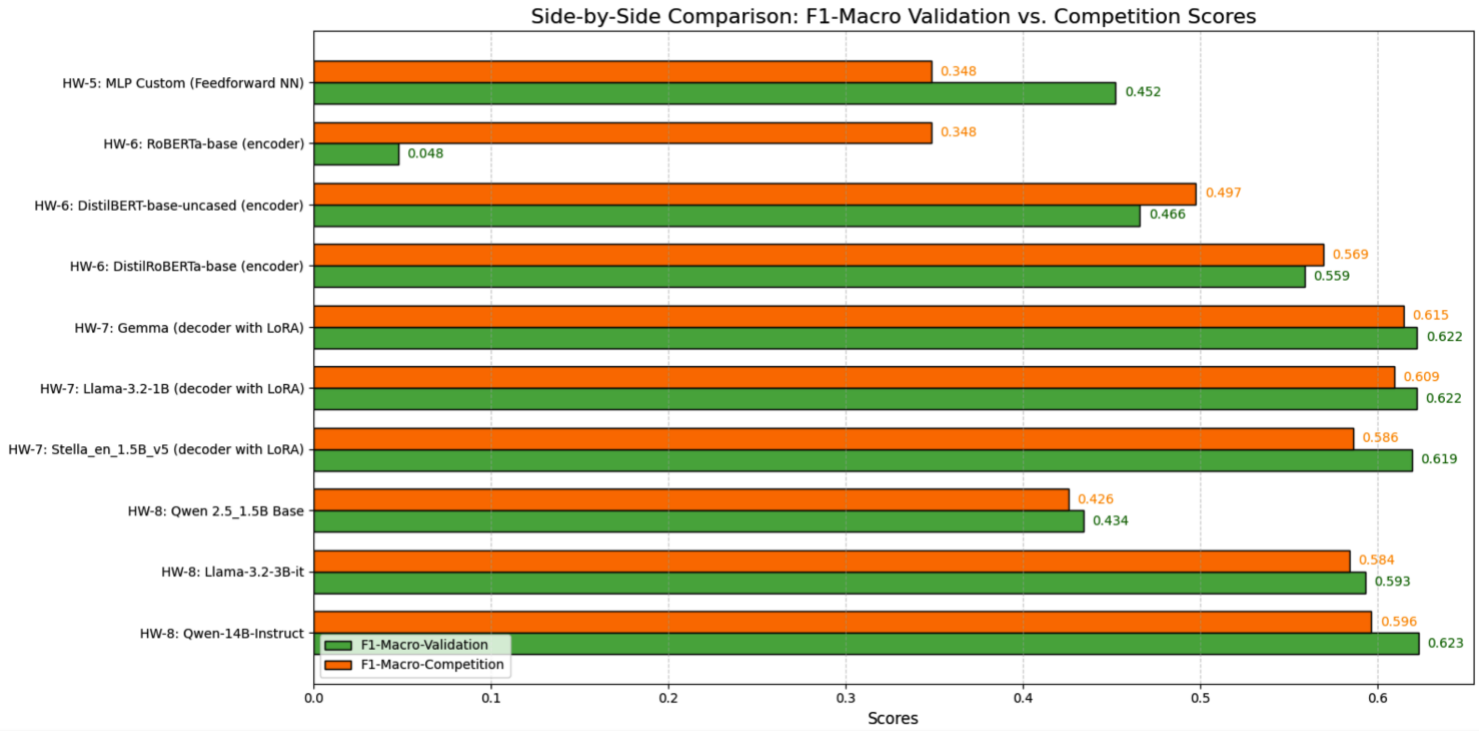
Report Introduction

The following report entails my analysis and results of a comprehensive learning journey of training different types of text classification models for a multilabel classification problem on a dataset of tweets. The primary goal was to train the models to be able to detect emotional sentiment and label tweets based on if the sentiment was present on a document level or not. I trained 10 different models on this dataset and made submissions to a Kaggle competition for each. The types of models used range from a classic Feed Forward Neural Network to a final Instruction Tuned 14 billion parameter model that I trained using the RunPod platform. All of the models with the exception of the last were trained using Google Colabs pro subscription using A100 GPUS. The final model was trained using an A100 PCIe GPU where I had to fully set up my own environment with the relevant packages and dependencies. I also had to ensure the large model was stored in the correct location. Throughout the report I will give full insights into the training process for each of these models to include: Model Results, Descriptions, Data Splits, Hyperparameters used, and future recommendations and thoughts on the model results.

Full Results

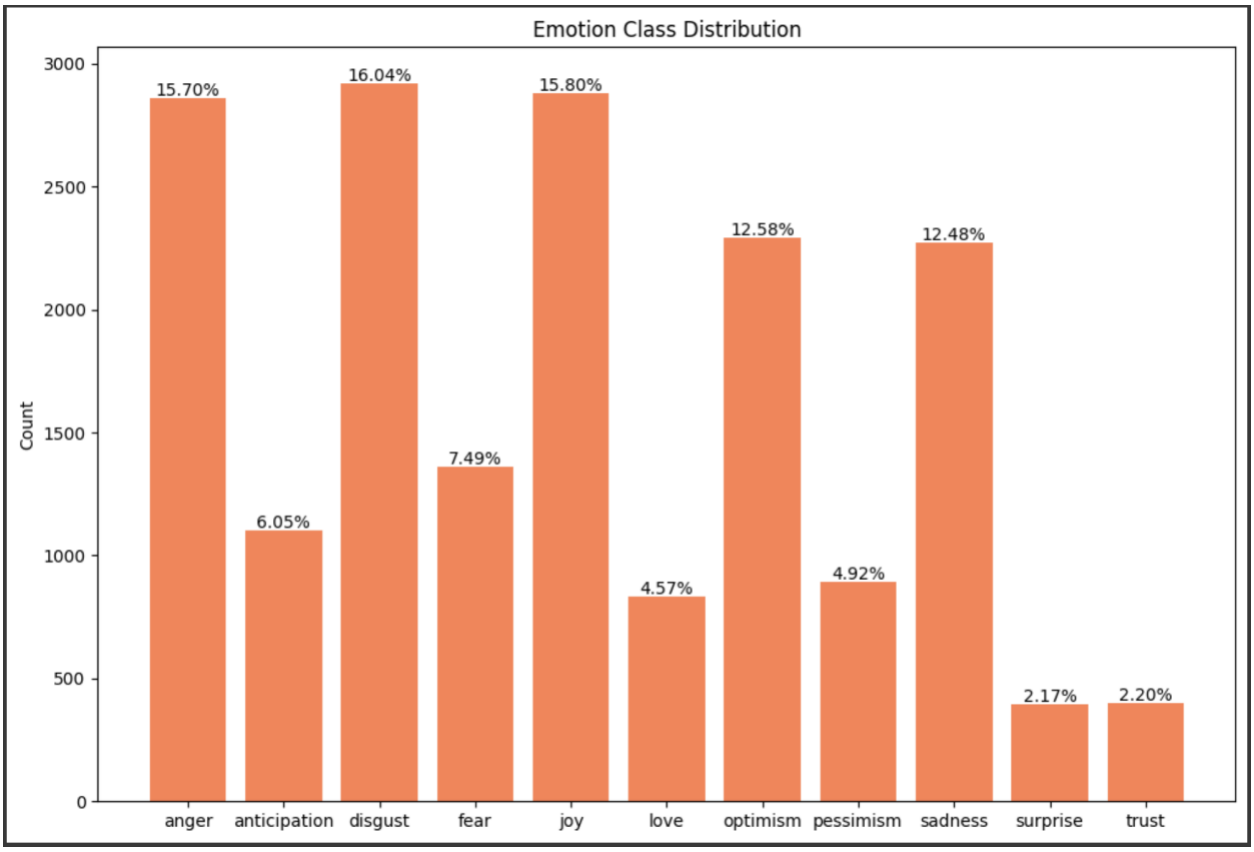
Figure 1 below is where I will begin my analysis. It shows a comprehensive comparison of each model's F1 Macro score on both the validation set and on the Kaggle test set.

Figure 1:



F1 Macro was the given metric for the competition since the dataset was highly imbalanced across the labels. As shown in the plot, the best model was the Gemma-2B base decoder model with a final validation score of 0.622 and competition score of just below 0.615. The worst performing model was my initial large language model training using a RoBERTa base decoder where I had not yet accounted for the class imbalance properly during the model training phase. Class weighting played a pivotal role in obtaining better results. Below in figure 2 we can see the class imbalance among each of the emotional sentiment classes present in the training set.

Figure 2:



Now I will proceed to break down each model and present a full analysis.

HW5: FFNN

1. **Feed-Forward Neural Network (FFNN) – Validation F1-Macro: 0.45215, Kaggle F1-Macro: 0.34815**
    - **Description**: A multilayer perceptron with multiple embedding layers and hidden layers to support non-linear detections.
    - **Splits:** For this model I used a standard split on the training dataset without accounting for the class imbalance. I split the data using 70% for training and 30% for validation.
    - **Hyperparamters:**
        1. **Epochs: 10**
        2. **Batch Size Train: 64**
        3. **Batch Size Eval: 128**
        4. **Weight Decay: 0.01**
        5. **Learning Rate: 0.005**
        6. **Optimizer: AdamW**
        7. **Learning Rate Scheduler: Reduce_LR_on_Plateau**

    - For this model I did not run any class specific metrics because I was just attempting to get my f1 score as high as possible. After the first 5 epochs I altered my hyperparameters to include the learning rate scheduler and began training again altering the number of epochs and starting from the best checkpoint.
    - **In the future:** I would adjust the model to account correctly for the class imbalance and incorporate a Multi Label Stratified Kfold cross validation for the splits to ensure equal distribution of the classes across the training and validation sets.

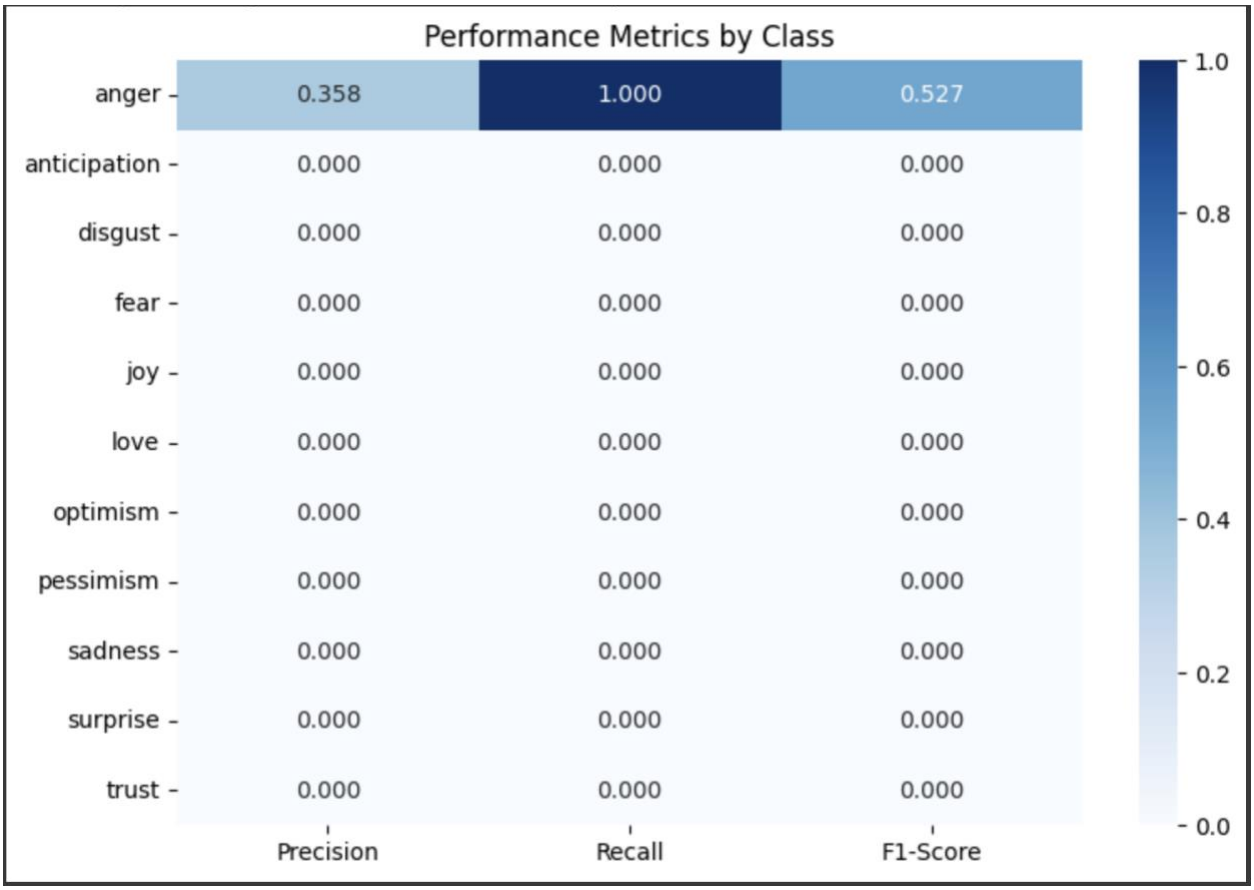
HW6: Encoder Models

2. **RoBERTa-base (encoder) - Validation F1-Macro: 0.0479, Kaggle F1-Macro: 0.3482**
    - **Description**: A robust encoder-only model optimized for text classification tasks.
    - **Key Features**: Struggled to predict the most underrepresented classes and the scores reflected this. All the predictions consisted of only three classes on the final training set: ‘anger’, ‘disgust’, or ‘joy’.
    - **Splits:** For this model I used a standard split on the training dataset without accounting for the class imbalance. I split the data using 70% for training and 30% for validation. I tried to implement class weights for this model to fine tune the f1 score but was unsuccessful. So, I stayed with the base model and no class weight implementation thus explaining the lower f1-score.
    - **Hyperparamters:**
        1. **Epochs: 2**
        2. **Batch Size Train: 16**

3. **Batch Size Eval: 16**
4. **Weight Decay: 0.01**
5. **Learning Rate: 0.0005**
6. **Optimizer: AdamW**
7. **Learning Rate Scheduler: Reduce_LR_on_plateau**

- **In the future:** I would adjust the model to account correctly for the class imbalance and incorporate a Multi Label Stratified Kfold cross validation for the splits to ensure equal distribution of the classes across the training and validation sets.

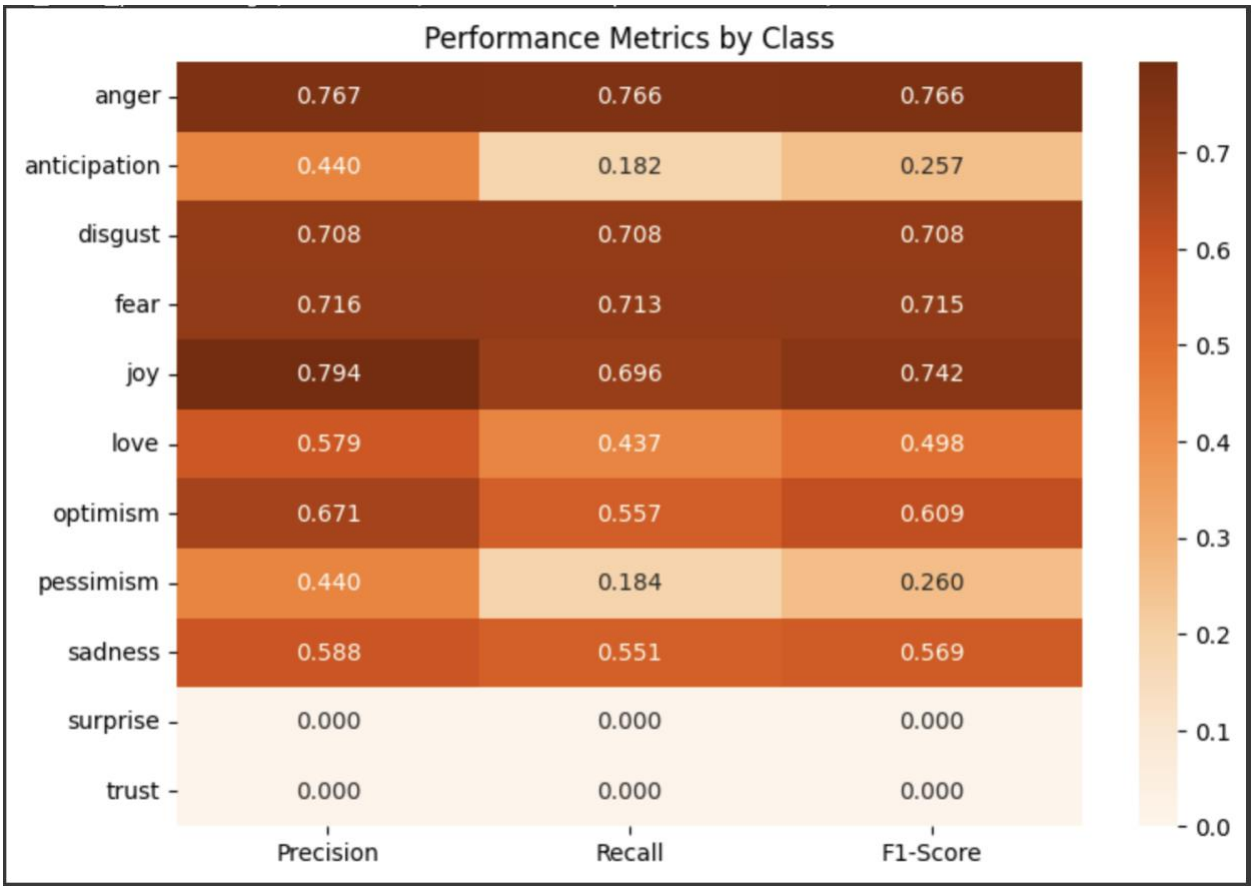
Figure 3: Validation Results Among Classes – Roberta-Base



3. **DistilBERT-base-uncased (encoder) - Validation F1-Macro: 0.4658, Kaggle F1-Macro: 0.49747**
   - **Description**: A distilled version of BERT, providing a lightweight and efficient encoder classification model.
   - **Key Features**: Improvement in the overall macro and small improvement in the more minor classes. Still no improvement in the most underrepresented classes though (surprise and trust).

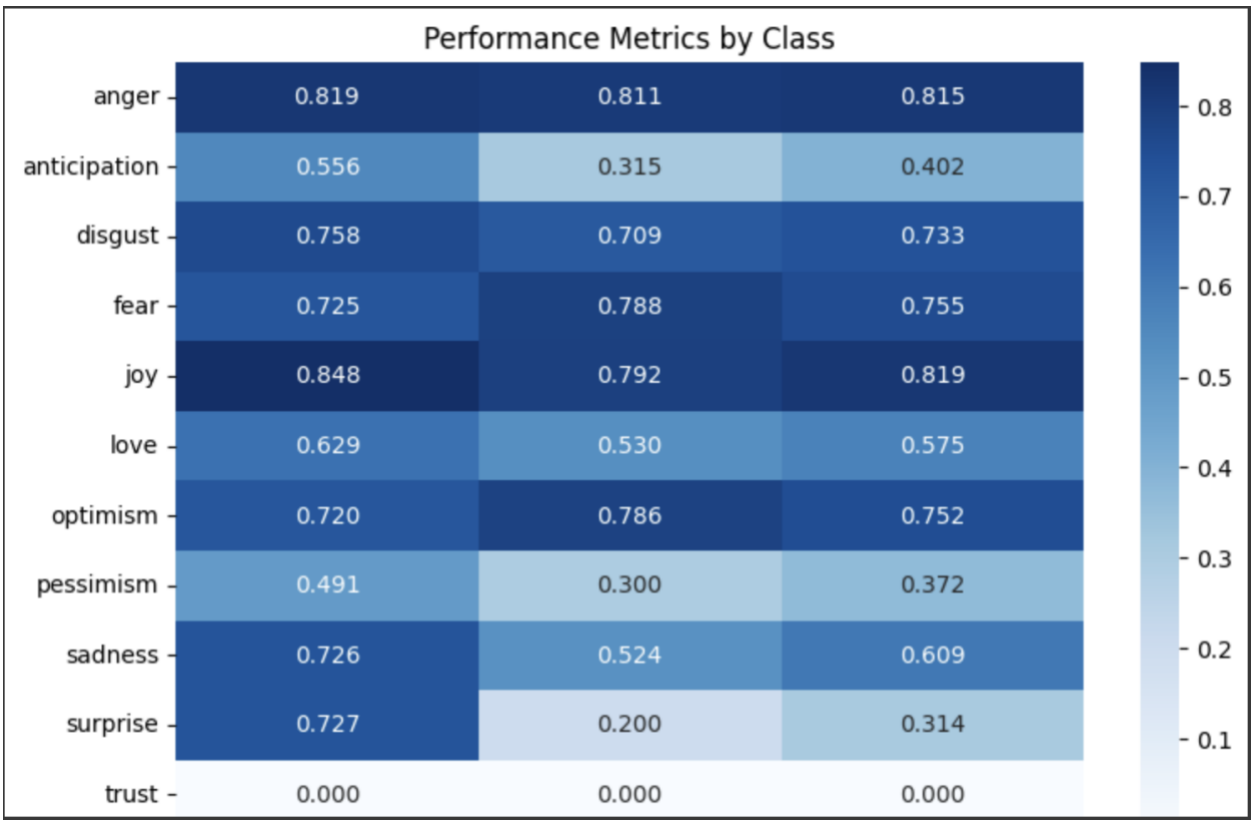
- **Splits**: I did some research on the best way to account for class imbalance when splitting a multilabel dataset and I came upon the MultiLabelStratifiedKFold (MKF) splitter that could use cross validation to correctly split the data based on the class distributions. I used 10 folds here and was able to get really good results for the splits that accurately represented the full training class distribution. I also changed the learning rate scheduler to linear because I tried both and the results improved with the change.
- **Hyperparamters:**
    1. **Epochs: 5**
    2. **Batch Size Train: 16**
    3. **Batch Size Eval: 16**
    4. **Weight Decay: 0.01**
    5. **Learning Rate: 0.0005**
    6. **Optimizer: AdamW**
    7. **Learning Rate Scheduler: Linear**
- **In the future:** I would experiment with different cross validation strategies to see which number of splits returned the best validation results. I would also integrate proper threshold finetuning to imprve the cutoff threshold for a prediction being present or not. The one I used for this model was generalized for the entire dataset. I set it to 0.4.

Figure 4: Validation Results Among Classes – DistilBert-Base-Uncased



4. **DistilRoBERTa-base (encoder) - Validation F1-Macro: 0.5587, Kaggle F1-Macro: 0.5691**
    - **Description**: A distilled version of RoBERTa, keeping both the lightweight efficiency of a distilled model and the robustness of RoBERTa.
    - **Key Features**: Overall, the best performing model among my encoder specific models. This model was finally able to correctly predict some of the most underrepresented labels in the validation set for the surprise label. However, this model was unable to correctly identify trust.
    - **Splits**: As with the DistilBERT model I used the stratified split with 10 folds. I was also able to correctly represent the class weights and integrate them correctly to improve my model. I independently weighed each class relative to that class instead of trying to get the weights across all labels within the dataset. I used a specific threshold for the logits for each class to ensure that the cutoff for prediction of a class was optimal to improve the f1 score.
    - **Hyperparamters:**
        1. **Epochs: 5**
        2. **Batch Size Train: 16**
        3. **Batch Size Eval: 16**
        4. **Weight Decay: 0.01**
        5. **Learning Rate: 0.0005**
        6. **Optimizer: AdamW**
        7. **Learning Rate Scheduler: Linear**
    - **In the future:** I would experiment with different cross validation strategies and better fine tune the threshold I used to cut off the classification of a class being present or not for the labels.

Figure 5: Validation Results Among Classes – DistilRoberta – Base

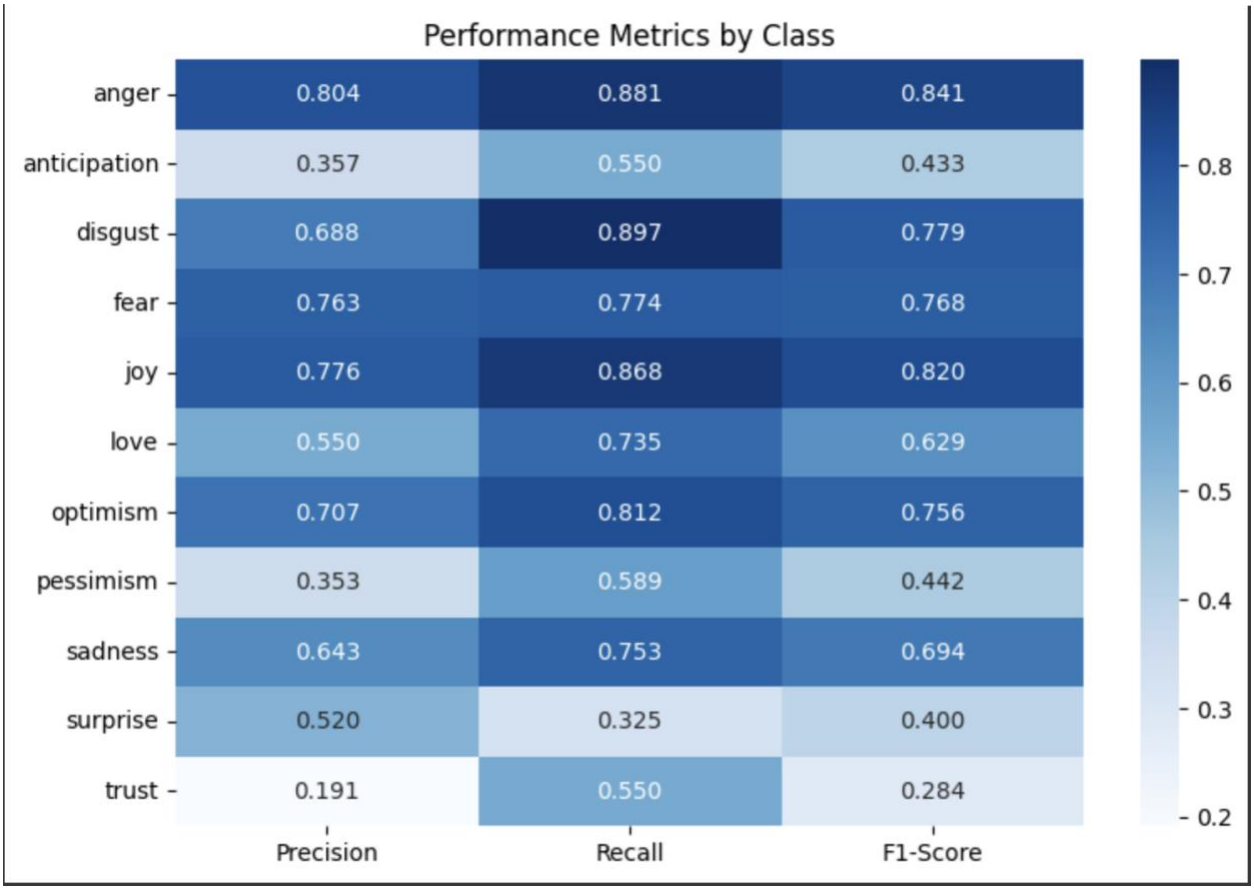


HW7: Decoder Models (LoRA)

5. **Gemma-2B-Base (decoder with QLoRA) - Validation F1-Macro: 0.6222, Kaggle F1-Macro: 0.6145**
    - **Description**: A decoder model that I finetuned using Quantized Low Rank Adaptation (QLoRA). This allowed me to download the full model and modify the embedding weights to fit on the GPU.
    - **Key Features**: This model was my best performing model in terms of F1 score. It was finally able to correctly predict some instances in both most underrepresented classes. The final model weights used to do inferencing may be overtrained due to the length of my training process and the validation and training losses diverging slightly. It may not end up being my overall best when the final datasets are released.
    - **Splits**: Again, I used the MKF splitting process using 10 folds to split the dataset. I was also able to correctly represent the class weights and integrate them correctly to improve my model. I independently weighed each class relative to that class instead of trying to get the weights across all labels within the dataset. I used a specific threshold for the logits for each class to ensure that the cutoff for prediction of a class was optimal to improve the f1 score.

- **Hyperparameters**:
    1. **Epochs: 5**
    2. **Batch Size Train: 16**
    3. **Batch Size Eval: 16**
    4. **Weight Decay: 0.01**
    5. **Learning Rate: 0.0005**
    6. **Optimizer: AdamW**
    7. **Learning Rate Scheduler: Not Used**
- **In the future:** I would experiment with different cross validation strategies and experiment with different hyperparameters or learning rate schedulers to improve model performance.

Figure 6: Validation Results Among Classes – Gemma-2B-Base



6. **Llama-3.2-1B (decoder with LoRA) - Validation F1-Macro: 0.6217, Kaggle F1-Macro: 0.6091**

- **Description**: This model was also like the Gemma model as a base classification model using QLORA.
- **Key Features**: Improved performance on the most underrepresented classes in comparison to the standard encoder models. However, it failed to achieve an F1 score that surpassed that of the Gemma model.
- **Splits**: Again, I used the MKF splitting process using 10 folds to split the dataset. I was also able to correctly represent the class weights and integrate them correctly to improve my model. I independently weighed each class relative to that class instead of trying to get the weights across all labels within the dataset. I used a specific threshold for the logits for each class to ensure that the cutoff for prediction of a class was optimal to improve the f1 score.
- **Hyperparameters**:
    1. **Epochs: 3**
    2. **Batch Size Train: 16**
    3. **Batch Size Eval: 16**
    4. **Weight Decay: 0.01**
    5. **Learning Rate: 0.0005**
    6. **Optimizer: AdamW**
    7. **Learning Rate Scheduler: Linear**
- **In the future:** I would experiment with different cross validation strategies and experiment with different hyperparameters or learning rate schedulers to improve model performance.

Figure 7: Validation Results Among Classes – Llama-3.2-1B

7. **Stella_en_1.5B_v5 (decoder with LoRA) - Validation F1-Macro: 0.6191, Kaggle F1-Macro: 0.5860**
    - **Description**: Another powerful decoder model where I used LORA for training.
    - **Key Features**: Balanced performance achieved. Performs better on ‘trust’ than both the Gemma and Llama models.
    - **Splits**: Again, I used the MKF splitting process to split the dataset. This time, however, I experimented with using 15 folds to see how well the model performed. I was also able to correctly represent the class weights and integrate them correctly to improve my model. I independently weighed each class relative to that class instead of trying to get the weights across all labels within the dataset. I used a specific threshold for the logits for each class to ensure that the cutoff forprediction of a class was optimal to improve the f1 score.
    - **Important Note:** This model was the most difficult to train of these base decoder model because it had a different end of sequence token and pad token, and I was just learning how those changed among models. I also switched back to the ‘reduce_lr_on_plateau’ learning rate scheduler to test its performance against the other options I had tried before to see if there were any improvements. I anticipated that this would be my most well-rounded model because of the improved F1-macro on the validation set, but the competition score was not as high as I had hoped. Given that only half of the testing data was used, I may yet see this model rise to the top. It could be though, that since the validation data size

was so small there was not enough data to get a full glimpse of the trained weights predictive power on more data.

- **Hyperparameters:**
    1. **Epochs: 4**
    2. **Batch Size Train: 16**
    3. **Batch Size Eval: 16**
    4. **Weight Decay: 0.01**
    5. **Learning Rate: 0.0005**
    6. **Optimizer: AdamW**
    7. **Learning Rate Scheduler: Reduce_lr_on_Plateau**
- **In the future:** I would experiment with different cross validation strategies and experiment with different hyperparameters or learning rate schedulers to improve model performance.

Figure 8: Validation Results Among Classes – Stella-en-1.5B-v5

HW8: Advanced Decoder Models

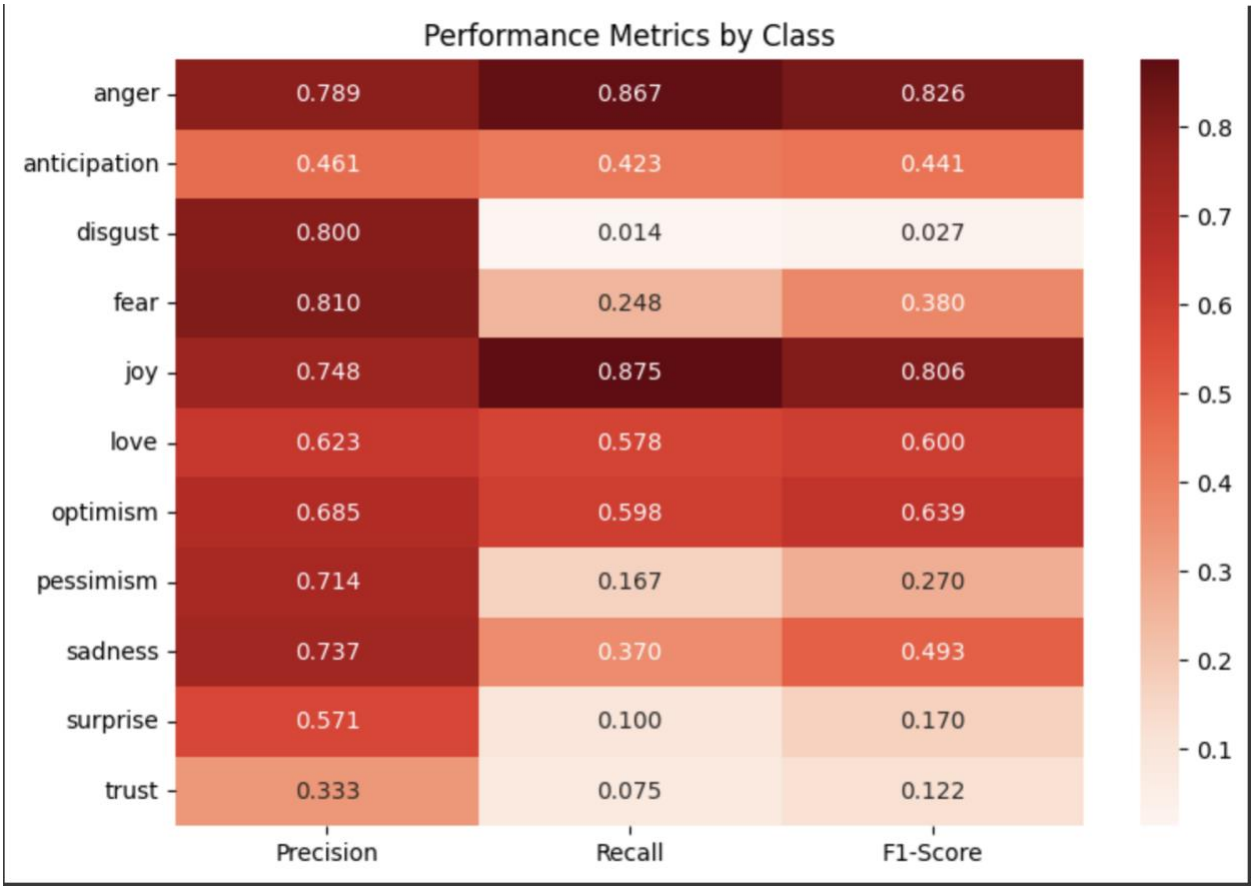
8. **Llama-3.2-3B-it (decoder) - Validation F1-Macro: 0.5929, Kaggle F1-Macro: 0.58405**
    - **Description**: A decoder model leveraging iterative fine-tuning and enhanced cross-validation techniques to achieve balanced performance across frequent and minority emotion classes.
    - **Key Features**: Significant improvements in recall for "disgust" and "trust" but results for F1 Macro and among the classes still lagged behind my prior models.
    - **Splits:** Again, I used the MKF splitting process to split the dataset. This time, however, I experimented with using 3 folds to see how well the model performed.
    - **Important Note:** I tried training this model without a reference as to how the tokenizer integrated the pad token. I faced many issues when loading the model to train and when attempting to reload the model to perform inference. This model doesn't have a set padding token so I first attempted to set my own but the embedding sizes and vocab size for the tokenizer became mismatched. I even tried to force a match for the lengths of both the tokenizer vocab and embedding lengths but still faced issues with my model not generating labels. I later realized that I needed to change the padding side for the model to generate labels and needed to use a specific pad token to correctly pad the dataset and perform training and inference. I think I learned the most when training this model because I did most of the training without any references. I learned a lot about the chat formatting and cleaning of outputted generations.
    - **Hyperparameters:**
        1. **Epochs: 3**
        2. **Batch Size Train: 16**
        3. **Batch Size Eval: 16**
        4. **Weight Decay: 0.01**
        5. **Learning Rate: 0.0005**
        6. **Optimizer: AdamW**
        7. **Learning Rate Scheduler: Not Used**
    - **In the future:** I would experiment with different cross validation strategies and experiment with different hyperparameters or learning rate schedulers to improve model performance.

Figure 9: Validation Among Classes – Llama-3.2-3B-it

9. **Qwen 2.5_1.5B Base (decoder) - Validation F1-Macro: 0.4341, Kaggle F1-Macro: 0.42565**
    - **Description**: A decoder model trained to balance performance on frequent and sparse emotion labels, with specific focus on improving generalization and handling of underrepresented classes.
    - **Key Features**: Precision was strong, but recall remained a challenge, the model did not perform as well here on F1 either because I may not have used the best prompting techniques to format the chat.
    - **Splits:** Again, I used the MKF splitting process to split the dataset. This time I resumed using 10 folds for the split since this number had remained my best selection so far.
    - **Hyperparameters:**
        1. **Epochs: 3**
        2. **Batch Size Train: 16**
        3. **Batch Size Eval: 16**
        4. **Weight Decay: 0.01**
        5. **Learning Rate: 0.0005**
        6. **Optimizer: AdamW**
        7. **Learning Rate Scheduler: Not Used**
    - **In the future:** I would experiment with different cross validation strategies and experiment with different hyperparameters or learning rate schedulers to improve

model performance. I would also try to figure out how to implement class weights into the model to see if it will improve the performance.

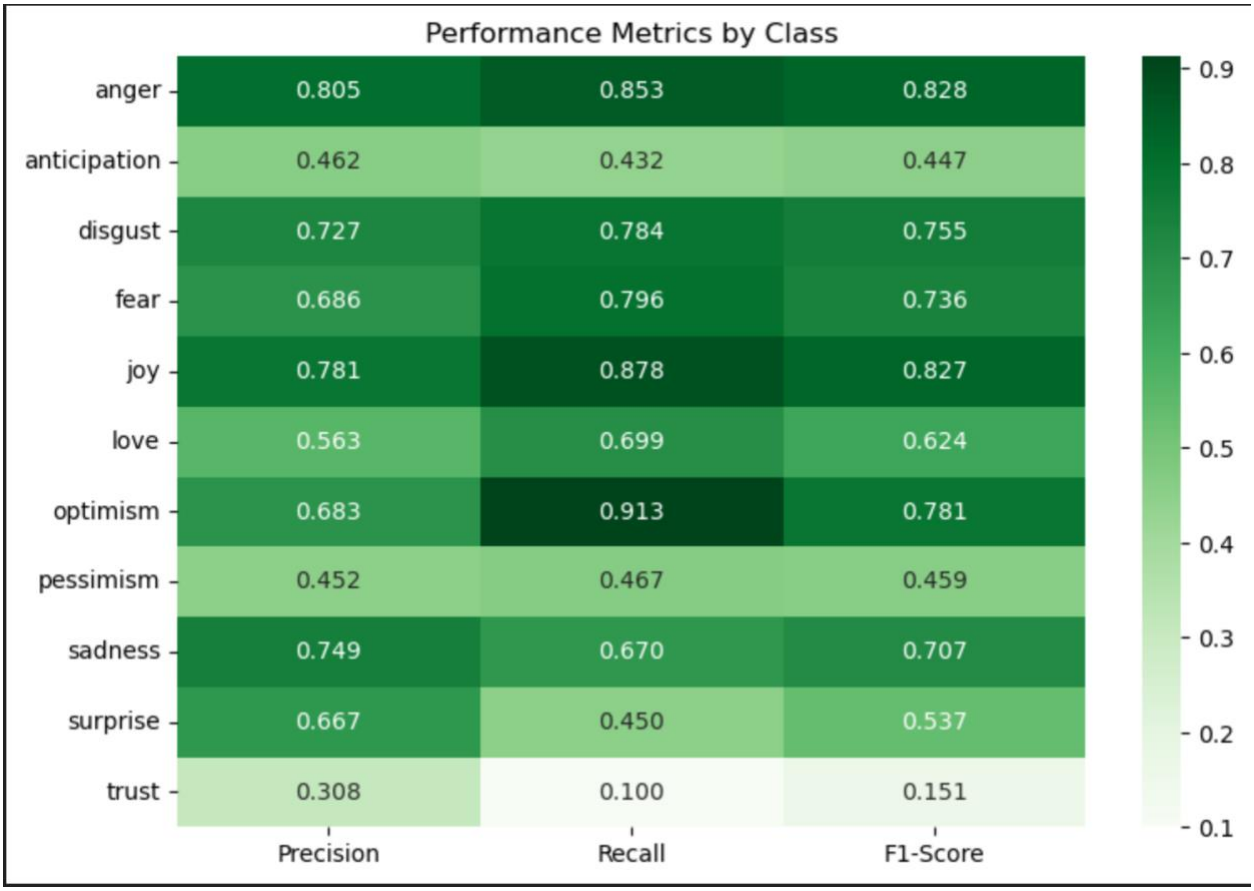
Figure 10: Validation Results Among Classes – Qwen-2.5-1.5B-Base



10. **Qwen-14B-Instruct (decoder) - Validation F1-Macro: 0.6229, Kaggle F1-Macro: 0.5963**
    - **Description**: An instruction-tuned decoder model designed to understand nuanced inter-label relationships, achieving a strong balance of precision and recall across frequent and sparse emotion classes.
    - **Key Features**: Achieved the best F1-Macro among all of my models on the validation set. However when I submitted the predictions on the test set they were less than I expected. Again though, since the test set is ran only against half of the actual test predictions in Kaggle, this model may end up performing better than all of my other models.
    - **Splits:** Again, I used the MKF splitting process to split the dataset. This time I resumed using 10 folds for the split since this number had remained my best selection so far.

- **Important Note:** I initially tried to train a 72 billion parameter model using RunPod during this training experiment. However, I realized that the model was too big to hold both the pytorch tensors for the embeddings and the model. I incurred many out of memory errors for this training process and was unsuccessful. So, I opted to train this smaller model. I had to set my own entire environment and select the appropriate storage for both the huggingface cache for the initial model download and the pytorch storage cache. This was a great learning experience for me. I also switched back to a linear learning rate scheduler to examine the impact on training and performance.
- **Hyperparameters:**
    1. **Epochs: 3**
    2. **Batch Size Train: 16**
    3. **Batch Size Eval: 16**
    4. **Weight Decay: 0.01**
    5. **Learning Rate: 0.0005**
    6. **Optimizer: AdamW**
    7. **Learning Rate Scheduler: Linear**
- **In the future:** I could attempt to incorporate class weights into the model and experiment with different splits to improve model performance. I also would like to reattempt the 72B model from QWEN to see if I can manage to incorporate multiple GPUs or a GPU with more memory in the training process. I think this is probably my next experiment that I will try on my own.

Figure 11: Validation Among Classes – Qwen-14B-Instruct


Performance Metrics by Class
anger
anticipation
disgust
fear
joy
love
optimism
pessimism
sadness
surprise
trust
0.805
0.853
0.828
0.462
0.432
0.447
0.727
0.784
0.755
0.686
0.796
0.736
0.781
0.878
0.827
0.563
0.699
0.624
0.683
0.913
0.781
0.452
0.467
0.459
0.749
0.670
0.707
0.667
0.450
0.537
0.308
0.100
0.151
Precision
Recall
F1-Score
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1