114062424 William

Data Mining Lab 2 Phase 3 Report

1. Model Development

1.1 Overview

For this task, I developed a complete training pipeline that combines 5-fold cross-validation, pseudo-labeling, and a final two-model ensemble. I used microsoft/deberta-v3-base as the backbone because it consistently provides the strongest performance among base-sized transformer models for emotion classification. The final pipeline consists of four major stages:

a. Text preprocessing and cleaning

b. Cross-validation training (CV5) to generate OOF logits

c. Pseudo-labeling high-confidence test samples

d. Final model trained on (train + pseudo-labels) and ensemble with CV5

This design lets me get both:

a. A reliable OOF evaluation (so I can check real generalization), and

b. A larger training set from pseudo-labeling (to push performance beyond what supervised data alone can achieve).

1.2 Data Cleaning & Preprocessing

I applied a lightweight but effective preprocessing pipeline to normalize social-media text:

a. Replace URLs → <url>

b. Replace usernames → <user>

c. Normalize character elongation (e.g., “soooo” → “sooo”)

d. Collapse repeated whitespace

e. Strip and lowercase

These rules help reduce noise while keeping emotional cues intact. Tokenization uses AutoTokenizer (DeBERTa-v3-base, fast tokenizer) with:

a. max_length = 192

b. No padding during mapping (padding handled dynamically by DataCollatorWithPadding)

1.3 5-Fold Cross-Validation (CV5)

I performed Stratified K-Fold (N=5) training on the full dataset. For each fold:

a. I loaded a fresh DeBERTa-v3-base model

b. Used consistent training settings:

- LR = 2e-5
- 4 epochs
- Cosine LR schedule
- Gradient accumulation (effective batch size = 16)
- Dropout increases (0.10) for better regularization
- FP16 for efficiency

c. And save some files:

- Out-of-fold (OOF) logits for evaluation
- Per-fold predictions on the test set

The test logits were averaged across folds:

cv5_test_logits = (sum of fold predictions) / 5

This gives a stable, low-variance base predictor and give result for OOF evaluation Weighted F1 ≈ 0.688. OOF is always lower than final performance because it's a strict honest evaluation.

1.4 Pseudo-Labeling with Confidence Threshold

I applied pseudo-labeling using the CV5 test-set predictions. Steps:

a. Convert CV5 logits to probabilities using softmax

b. Select samples where max_prob >= 0.90

c. Use predicted label as pseudo-label

d. Append these samples to the supervised training set

This created a larger hybrid dataset which is train combines with pseudo-labels. This helps the model learn patterns from test-domain examples. It also improves minority-class representation if the pseudo-labels are clean.

1.5 Final “Pseudo Model” Training

I trained one final model on the expanded dataset. I kept a small 10% validation split (stratified) to avoid leaking the pseudo-labels into evaluation. Then, training hyperparameters were the same as CV5 and validation performance improved slightly, especially in minority classes.

1.6 Final Two-Model Ensemble

To combine the strengths of both models, I used a simple but effective linear ensemble:

final_logits = 0.7 * cv5_test_logits + 0.3 * pseudo_test_logits

Using a heavier weight on CV5 helps prevent pseudo-label noise from dominating.

1.7 Result Summary

On the test set:

a. **Final F1 Weighted ≈ 0.7023**
b. Stable across several submissions
c. Ensemble outperformed any single model and outperformed pseudo-model alone

The pipeline successfully balances:

a. Generalization (via CV5)
b. Larger dataset (via pseudo-labels)
c. Robust predictions (via ensemble)

2. Bonus Section

This section explains the many techniques I tried earlier, why some of them did not work well, and what I learned from the process. I write this from a personal, reflective perspective.

2.1 Oversampling & Class-Balanced Tricks

I tried several imbalance-handling techniques:

a. Random oversampling
b. Class-balanced loss
c. Focal loss
d. Label smoothing
e. Higher-γ focal tuning
f. Combination of all of the above

Why it didn't work well

Oversampling made the minority classes appear too frequently, causing the model to hallucinate those emotions in normal contexts. Focal loss helped slightly but made optimization harder, especially with DeBERTa's deep architecture. Some runs actually hurt the majority classes and reduced weighted F1.

Insight

Emotion classification has nuanced boundaries—forcing minority classes too aggressively harms overall performance. Balanced improvements require architecture-level gains, not brute-force resampling.

2.2 Gradient Checkpointing + DeBERTa-v3-Large (OOM Issues)

I experimented with DeBERTa-v3-Large for higher representational power. Even with these parameters:

a. batch size = 1
b. gradient accumulation
c. gradient checkpointing
d. FP16

I kept hitting problems:

a. CUDA OOM
b. Compute graph backward errors
c. Fragmentation errors during training

Why it didn't work

10.8 GB GPU wasn't enough for a stable training run. Some checkpoints were unstable and crushed training halfway. Even when it trained, the improvement was tiny compared to the cost.

Insight

Bigger models are not automatically better. For this dataset and GPU size, DeBERTa-v3-base is the real sweet spot.

2.3 Multi-model Ensemble of Random Seeds (BERTweet, DistilRoberta, etc.)

I previously trained multiple single models:

a. BERTweet-base

b. BERTweet-large
c. DistilRoBERTa
d. Twitter-RoBERTa
e. RoBERTa-Large MNLI

I attempted to ensemble them with:

a. simple averaging
b. bias tuning
c. weighted averaging

Why it didn't work

Some models were significantly weaker. When combining strong models (DeBERTa) with weaker ones, the ensemble moved toward the average, not upward. A few specialty models introduced systematic bias (e.g., overpredicting joy or anger), reducing stability.

Insight

Only models with competitive OOF performance should be part of an ensemble.
Averaging strong + weak = weaker.

2.4 Augmentation (EDA, Backtranslation, Random Swap/Drop)

I tried textual augmentations:

1. EDA (synonym replacement, random swap, random delete)
2. Character-level noise

Why it failed

Augmentation made emotional tone less natural. Replacing words changed the emotional direction of tweets. And also some augmentations shifted "anger" sentences into neutral tone or vice versa.

Insight

Emotion classification is extremely sensitive to wording. Augmentations designed for sentiment or binary tasks don't generalize well here.

2.5 Pure Pseudo-Label Model (without CV5)

I tried training a pseudo-label model without CV5, using just the single model's predictions.

Why it didn't work

The pseudo-labels were noisier because the base model wasn't strong enough. Training on its own mistakes amplified the noise and reduced performance on the real test set.

Insight

Pseudo-labeling only works well after obtaining a strong model with CV5. Otherwise, the pseudo-dataset becomes self-reinforcing noise.

2.6 Bias Tuning & Logit Calibration

I tried adjusting output logits by applying per-class bias vectors. On validation, I could push weighted F1 up to 0.71+ using bias tuning.

Why it didn't work on test

The test distribution does not perfectly match the validation distribution. Over-optimizing bias for a small validation set caused overfitting. Biased models looked great on validation but dropped on test to ~0.697.

Insight

Calibration must be based on OOF, not a small validation split. Logit biasing must be conservative.

2.7 Learning Rate Schedules, Seed Sweeps, Longer Training

I tried:

a. 8–10 epoch training
b. Linear warmup/decay
c. Cosine with restarts
d. Higher learning rates

Why it didn't help

More epochs pushed the model into overfitting. Seed sweeps gave some variation but not enough to justify multiple runs. LR schedules gave diminishing returns.

Insight

This dataset prefers stable, moderate-length training (3–4 epochs). Most gains come from data, not hyper-tricks.