Sentiment Analysis of Tweets

STAT 362-0 Final Project

Jimmy Zhang

Problem and Motivation

- Natural Language Processing (NLP)
  - Multiclass Classification (13 classes)
- Applications
  - Social media moderation
  - Hate speech detection

Dataset Overview

- Kaggle Dataset
- 40,000 Observations
  - Tweet Text (feature)
  - Sentiment Label (target)
  - Tweet/User ID (unused)
- Word Embeddings with SentenceTransformer
  - Sentence Tokenization
  - Sequence Generation

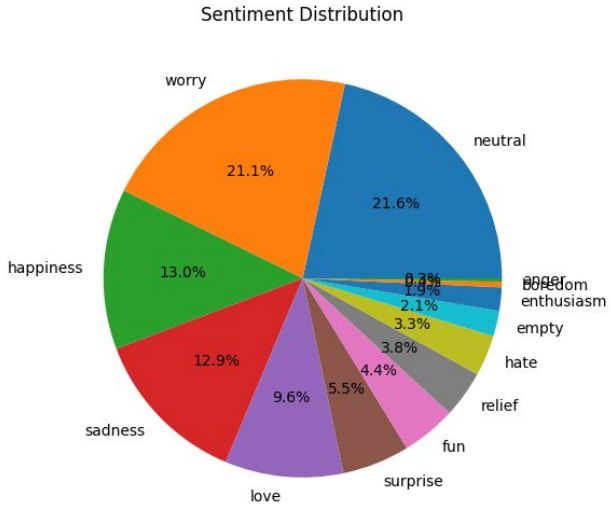

Figure 1. Distribution of sentiment classes in the dataset (13 classes).

Deep Learning Approach

- Pre-trained Embedding Models
  - Complexity vs Computational Cost tradeoff
- Baseline Models
  - Multilayer Perceptron (MLP)
  - Gradient Boosted Tree (XGBoost)
- Recurrent Neural Networks (RNNs) for Sequence Processing
  - Vanilla RNN
  - Gated RNN (LSTM, GRU)
  - Bidirectional RNN
- Transformer Model
- Different Architectures (Regularization, Dropout, etc.)

Experimental Setup

- Training/Validation/Test Sets (60/20/20 ratio)
- Compare 4 variants per model type
  - Select the best from each for overall analysis
- Multiclass Evaluation Metrics
  - Loss Function: Cross-Entropy
  - Final Performance: Accuracy
- Implementation
  - Google Colab: T4 GPU (High-RAM)
  - Local VSCode Poetry Environment

Key Results

- Training/Validation Loss Curves
- Model Ranking on Testing Set
- Confusion Matrix

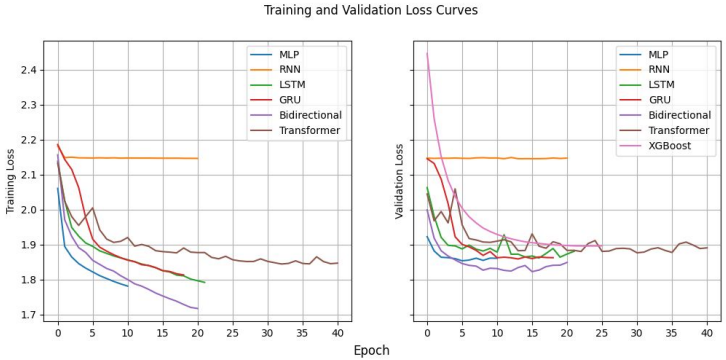

Figure 2. Final training and validation loss curves for the best variant of each model type.

	Model	Test Accuracy	Test Loss
0	Bidirectional	0.368	1.824
1	LSTM	0.364	1.865
2	MLP	0.361	1.858
3	GRU	0.359	1.863
4	Transformer	0.359	1.885
5	XGBoost	0.348	N/A
6	RNN	0.216	2.145

Table 1. Testing set performance of best models.

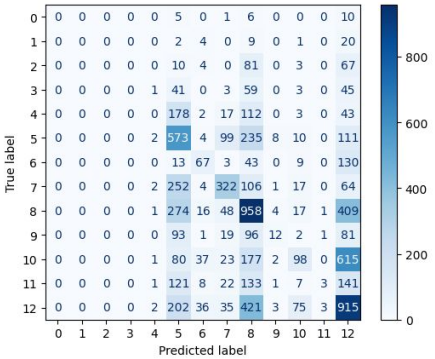

Figure 3. Confusion matrix from Bidirectional RNN predictions on testing set.

Conclusions

- Inherently a difficult task
  - Many classes with imbalance
  - Relatively limited training data compared to the real world
- Complexity doesn’t necessarily improve performance
  - Simpler architectures learned more quickly
  - Use Dropout/Regularization to reduce overfitting
- Deep learning approach was moderately successful
  - Small but clear improvement upon baseline

Reflection

- I expected the workflow with running code in Google Colab and exporting to VSCode to be tricky, but that part worked smoothly
- The biggest challenge was figuring out what different architectures I should try for each model type, since I wanted to make changes that would at least give me insightful results, if not performance improvements
- My original planned approach worked successfully overall
- Lessons:
    - In the experimental setup, sometimes I just need to pick an arbitrary starting point for my first model before making changes later
    - Maintaining an organized workflow is very important to avoid accidental confusion and obtaining non-reproducible results

AI Disclosure

- Gemini autocomplete in Google Colab
  - Improve efficiency when reproducing large chunks of code with only minor changes
  - I did not accept suggestions unless I knew exactly what that code would do and it matched with what I had previously done
  - If I ran into issues, I would carefully debug step-by-step
- VSCode GitHub Copilot
  - Generate Docstrings for some of the custom-defined functions

References

Chen, T. and Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785-794. http://dx.doi.org/10.1145/2939672.2939785.

Chollet, F., et al. (2015). Keras. https://keras.io/.

Reimers, N. and Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *arXiv:1908.10084*. https://arxiv.org/abs/1908.10084.

Valarmathi, B., et al. (2024). Sentiment Analysis of Covid-19 Twitter Data using Deep Learning Algorithm. *Procedia Computer Science, 235*, 3397-3407. https://doi.org/10.1016/j.procs.2024.04.320.