DA5401-2025 Data Challenge Report

Metric Learning to Study the Fitness of a Test Conversation Against an Evaluation Metric

Omkar Ashok Chaudhari
Roll No: NA22B059
Course: DA5401 – 2025 Data Challenge

November 21, 2025

Abstract

Contents

1 Introduction **2**

2 Problem Description **2**

3 Data Description **2**

4 Methodology **2**
4.1 Feature Engineering 2
4.2 Models 3
4.3 Blending Strategy 3

5 Experiments **3**
5.1 Training Setup 3
5.2 Ablation and Variants 3

6 Results **4**

7 Conclusion **4**

1 Introduction

- Conversational AI is used in many real applications, so we need a simple way to judge how good its responses are.
- Manually checking every answer is slow and not scalable.
- This challenge focuses on learning a model that can automatically score a response for a given evaluation metric.
- In this report, I briefly describe my approach, features, models, and final score on Kaggle.

2 Problem Description

- For each example, we are given a system prompt, a user prompt, a model response, and a metric name.
- The goal is to predict a fitness score between 0 and 10 that tells how well the response matches the intent of that metric.
- The true scores are produced by an LLM judge and are treated as ground truth.
- The competition metric is RMSE, so lower values mean better predictions.

3 Data Description

- The data comes in four main files: `train_data.json`, `test_data.json`, `metric_names.json`, and `metric_name_embeddings.npy`.
- Each train/test row has text fields: `system_prompt`, `user_prompt`, `response`, and a `metric_name`.
- The training set also has a discrete `score` from 0 to 10 as the target label.
- The metric embeddings file gives a fixed vector representation for each metric name, which is used as an input feature.

4 Methodology

4.1 Feature Engineering

- I first joined the three text fields into one string:
combined_text = system_prompt + user_prompt + response
.
- I converted this text into numeric features using TF–IDF, then reduced the size with Truncated SVD to get a fixed 256-dimensional text embedding for each row.
- For the metric side, I used the provided file [metric_name_embeddings.npy] to get a dense vector for each [metric_name].
- The final input feature for each example is a single vector formed by stacking:
features = (text_embedding, metric_embedding)
.

4.2 Models

- **Regression model**: I used [HistGradientBoostingRegressor] on the combined feature vector to directly predict scores in the 0–10 range.
- **Ordinal model**: I treated the score as ordered classes and trained 10 logistic models, one for each threshold [k = 1, 2, . . . , 10], to predict [P(score $\geq k$)].
- For the ordinal model, the final predicted score is the sum of these probabilities: pred_ord(x) = $\sum_{k=1}^{10} P(score \geq k \mid x)$

.

4.3 Blending Strategy

- Both models give a score between 0 and 10: [pred_reg(x)] from regression and [pred_ord(x)] from the ordinal model.
- I mixed them using a simple weighted average, giving more weight to the ordinal model: $\boxed{[score\_blend(x) = 0.6\times \text{ pred\_ord(x)} + 0.4 \times \text{ pred\_reg(x)}].}$
- Finally, I clipped the blended score to [0 $\leq$ score $\leq$ 10] and rounded to the nearest integer before creating the submission file.

5 Experiments

5.1 Training Setup

- All experiments were run inside `code.py` using the same feature set: TF–IDF + SVD text vectors and metric embeddings.
- The regression model was trained with 5-fold stratified cross-validation to keep the score distribution balanced.
- Each fold produced one model, and all models were averaged when predicting the final test outputs.
- The ordinal setup used 10 logistic models, each trained on the full dataset, one per threshold.
- All models were kept lightweight so they could train quickly and run without any GPU.

5.2 Ablation and Variants

- I tested using only the regression model and only the ordinal model to see which worked better individually.
- Regression alone gave smoother predictions but sometimes missed extreme scores.
- Ordinal classification handled the order of scores well but was slightly rigid in the middle ranges.
- Blending both models clearly improved stability and reduced RMSE compared to using a single model.
- I also tried changing SVD dimensions and TF–IDF limits, but the default settings gave consistently good results.

6 Results

- After testing all model variations, the blended approach (0.6 ordinal + 0.4 regression) gave the best performance.
- The final submission achieved an RMSE of **3.742** on the Kaggle private leaderboard.
- The score distribution looked balanced and matched the overall pattern seen in the training data.
- Both individual models were decent, but the blended prediction consistently performed better than either model alone.

7 Conclusion

- The challenge demonstrated that simple feature engineering and lightweight models can still work well for text-based evaluation tasks.
- Combining text embeddings with metric embeddings provided a meaningful way to link the prompt–response pair with the evaluation metric.
- Blending regression and ordinal models gave stable predictions and improved RMSE.
- Overall, the final approach was easy to run, fast to train, and produced a competitive leaderboard score.