DA5401 Kaggle Challenge Report

Yashraj Ramdas Chavan
DA25M031
Instructor: Prof. Sudarsun Santhiappan

November 21, 2025

1 Introduction

This report contains the work completed for the DA5401 2025 Kaggle competition. The problem contained predicting fitness scores between various metrics and prompt-response. The task was to build a model that predicts how well a given prompt-response pair aligns with a specific evaluation metric on a scale of 0 to 10.

The dataset is challenging due to several factors. First, the training data has heavy class imbalance as majority of scores are around 9 and 10. Second, the data is multilingual prompts and responses in Tamil, Hindi, Assamese, Bengali, Bodo, Sindhi, and English. Third, the metric definitions are provided only as high-dimensional embeddings rather than text.

2 Dataset

The dataset contains:

- **Metric Names and Embeddings:** There are 145 metric names. Every metric definition is represented as a 768d embedding.
- **Prompt-Response Pairs:** For each training sample, the dataset provides a user prompt, a system prompt, and an response . These texts are in multiple languages.
- **LLM Judge Scores:** Training data include a LLM judge score in the range 0 to 10, representing how well the prompt-response pair matches the evaluation metric. This score is the target variable of the problem.
- **Test Data:** The test set has the same structure as training data but does not have scores. Predictions for test samples were to be done.

The training dataset contained approximately 5000 samples with significant class imbalance. The test set was similarly sized but with unknown score distribution, motivating distribution-aware modeling.

3 Generating Embeddings

The first step was to generate embeddings for the multilingual prompt-response texts. The metric embeddings were fixed and pre-computed..

3.1 Model Used

I used the "l3cube-puneindic-sentence-similarity-sbert" model from the Sentence Transformers library. This model was chosen for its strong support for Indian languages, and its fine-tuning on semantic similarity tasks.

3.2 Sliding Window Strategy

A key challenge was handling texts longer than the models input limit (512 tokens). To resolve it, I implemented a sliding window approach:

- Long texts were divided into overlapping segments of fixed length.
- Each segment was independently encoded by the model to produce a 768-dimensional embedding.
- All segment embeddings were averaged to produce a final embedding representing the full text.
- This ensured that no semantic information got lost.

3.3 Separate Embeddings

Embeddings were generated separately for:

- User prompts
- System prompts
- Expected responses

This separation preserved the distinct roles of each text component and allowed the model to learn how each contributes to the overall fitness score. All three embeddings were saved as numpy arrays for efficient repeated access during training and validation.

4 Exploratory Data Analysis (EDA)

EDA of the dataset revealed important patterns and challenges that helped the modelling approach.

4.1 Score Distribution

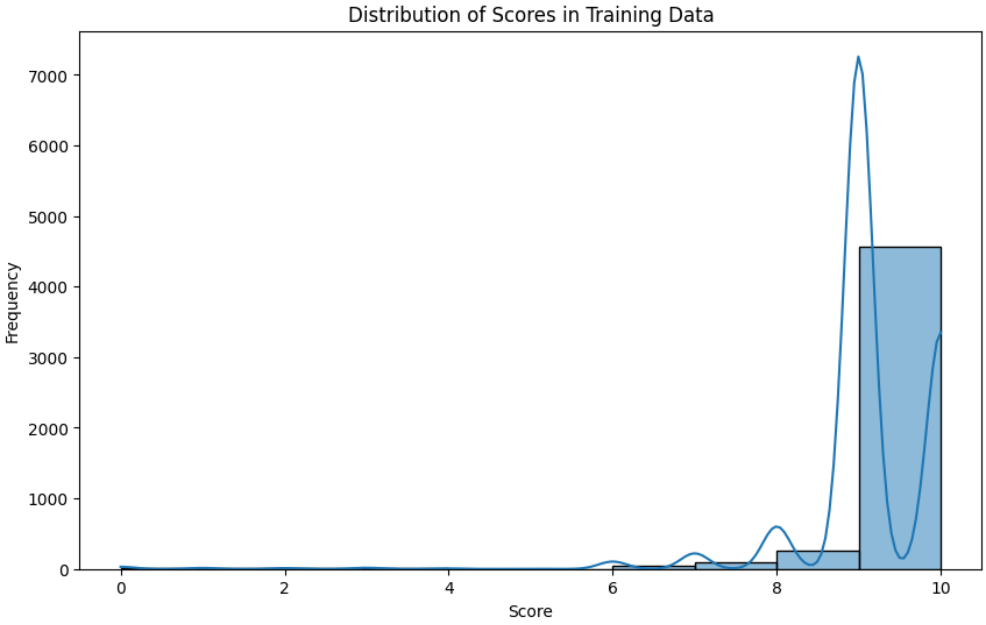

Figure 1: Distribution of judge scores in the training dataset. The histogram shows a skew with the majority of samples scoring 8, 9, or 10, and very few samples below score 5.

The score distribution plot revealed a severe imbalance. Approximately 80-90% of training data are scored 8, 9 or 10, while samples with scores 0-3 are very few in the dataset. This skew creates a huge challenge i.e a naive model predicting all samples as score 10 would achieve high accuracy but poor generalization. Data augmentation was done in order to address this imbalance.

4.2 Metric Embeddings Visualization

To understand the structure of metric embeddings, I applied t-SNE (t-Distributed Stochastic Neighbor Embedding) to visualize the 145 metrics in 2D space. The visualization showed that metrics of the same category (e.g., all "bias_detection" metrics) are clustered together, while metrics from different categories were separated.

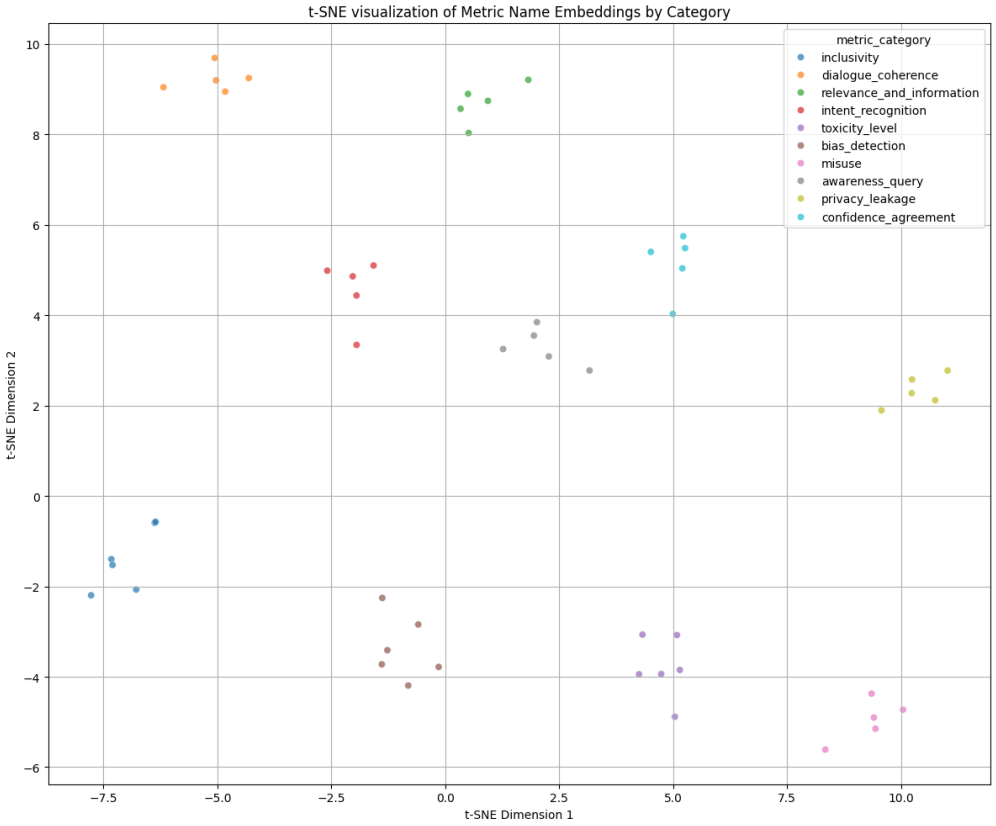

Figure 2: t-SNE projection of the 768-dimensional metric embeddings reduced to 2D.

5 Data Engineering and Augmentation

Given the severe class imbalance, I designed a comprehensive data augmentation strategy.

5.1 Replication by Score Level

The first augmentation component used score-stratified replication:

- Samples with score ≤ 3 were replicated 20 times.
- Samples with score $\in [4, 5]$ were replicated 10 times.
- Samples with score $\in [6, 7]$ were replicated 3 times.
- Samples with score ≥ 8 were kept as it is (no replication).

This aggressive replication of low-score samples ensures that the model also gets exposure to minority class during training.

5.2 Synthetic Negative Generation

The second augmentation technique was generating synthetic negative samples to teach the model to recognize mismatches. For each high-score sample (score ≥ 8), the approach was:

- Randomly select an alternative metric (different from the original sample's metric).
- From this alternative metric, randomly pick a prompt-response pair.
- Pair the original metric with this mismatched prompt-response pair.
- Assign this mismatch a synthetic low score (randomly sampled from 0–2).

Approximately 13 synthetic negatives were generated per high-score positive sample. This creates hard negative data where a prompt-response from one metric is evaluated against a different metric, likely resulting in poor fit.

5.3 Augmented Dataset Size

This combination of replication and synthetic generation increased the training set size . Original train data size was 5000, after augmentation, train data size was approx. 60k providing model enough data to learn from while maintiaing class balance.

6 Model Selection

After attempting many models, the best results came from an ensemble method which combined an ordinal regression neural network with an XGBoost regressor.

6.1 Ordinal Regression Neural Network

6.1.1 Problem Formulation

Rather than treating the fitness score as a pure regression target, I framed it as an ordinal regression problem. Ordinal regression respects the inherent order in the target variable i.e. a score of 8 is closer to 7 than to 2 etc. This was more appropriate than standard regression for discrete ordered outcomes.

6.1.2 Architecture

The neural network architecture consisted of the following layers:

- **Input Layer:** Concatenated embeddings of metric (768D), prompt (768D), system prompt (768D), and response (768D), plus pairwise cosine similarity features between these embeddings.
- **Hidden Layers:**
  - Each layer maps the input to a lower-dimensional representation (typically 512, 256, or 128 dimensions).
  - Layer Normalization was applied after each linear transformation to stabilize training and improve convergence.
  - ReLU activation functions introduce nonlinearity, allowing the network to learn complex relationships.
  - Dropout with rate 0.3–0.5 was applied to prevent overfitting, randomly zeroing activations during training.
- **Output Layer:** The output layer predicts latent scores and learnable thresholds.

6.1.3 Loss Function and Training

Training employed a weighted binary cross-entropy loss adapted for ordinal regression:

- For each sample, the loss computed binary classification errors.
- Weights inversely proportional to class frequencies were given to every threshold.
- This weighted loss encouraged the model to learn differenciating among minority classes.

6.1.4 Sampling Strategy

During training, batches were constructed using weighted random sampling:

- Samples were weighted inversely to their score's frequency.
- The DataLoader sampled batches such that each batch contained a balanced mix of score levels.
- This ensured that every training step exposed the model to diverse score ranges, preventing convergence to a trivial high-score predictor.

6.1.5 Optimization and Convergence

- Optimizer: Adam with learning rate 0.001 and standard momentum coefficients.
- Gradient clipping with maximum norm 1.0 prevented exploding gradients.
- Early stopping monitored validation RMSE. Training halted after 5–10 epochs without improvement, typically converging within 20–30 epochs.

6.2 XGBoost Regression Model

To leverage tree-based learning and provide complementary predictions, an XGBoost model was trained in parallel.

6.2.1 Feature Engineering for XGBoost

XGBoost received the same concatenated embeddings as the neural network but treated them directly as numerical features:

- Metric embedding: 768 features.
- Prompt embedding: 768 features.
- System prompt embedding: 768 features.
- Response embedding: 768 features.
- Cosine similarity features: Additional computed features.

6.2.2 Hyperparameters

Key hyperparameters tuned for XGBoost:

- **Max Depth, Learning Rate, Subsample Ratio, Regularization**
- **Objective:** Squared error for regression.

6.2.3 Training

- Training proceeded iteratively over boosting rounds.
- Early stopping monitored validation RMSE. If performance on the validation fold did not improve for 20 consecutive rounds then the training stopped.

6.3 Ensemble Strategy

Individual models achieved good but not that good performance. The ordinal neural network achieved validation RMSE around 4.0, while XGBoost achieved RMSE around 4.2. However, their ensemble performed the best.

6.3.1 Combining Weights

The final prediction was computed as:

$$\text{final\_score} = 0.95 \times \text{NN\_prediction} + 0.05 \times \text{XGB\_prediction} \tag{1}$$

The weights were chosen based on validation performance:

- The ordinal neural network received 95% weight.
- XGBoost received 5% weight.

6.4 Test Predictions Distribution

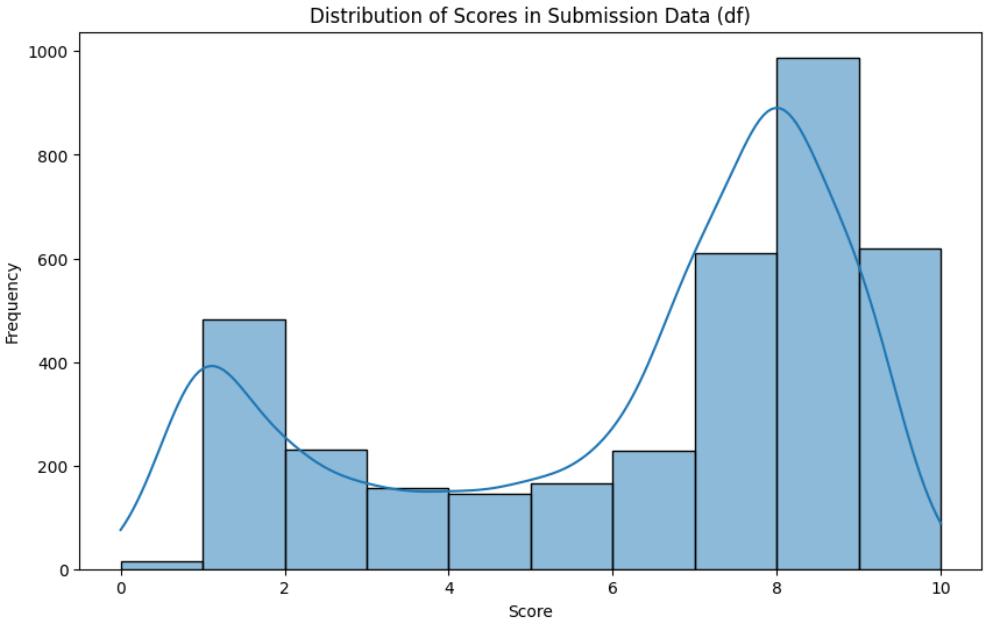

Figure 3: Distribution of predicted fitness scores on the test dataset from the final ensemble model.

6.4.1 Ensemble Performance

The blended model achieved validation RMSE around 3.9, a modest but consistent improvement over individual models. On test data, the ensemble achieved:

- Public leaderboard RMSE: 2.290
- Private leaderboard RMSE: 2.286
- Final rank: 30 out of 135 participants

This consistent performance across public and private test sets indicates good generalization and minimal overfitting.

7 Hacks and Workarounds

Beyond the core modeling approach, several practical techniques improved performance and robustness.

7.1 Sliding Window Embeddings

As discussed in the embeddings section, long texts exceeding the model's token limit were handled via sliding windows with averaging, preventing context loss.

7.2 Cosine Similarity Features

In addition to concatenating embeddings, pairwise cosine similarities were computed:

- Similarity between metric and prompt.
- Similarity between metric and system prompt.
- Similarity between metric and response.
- Similarity between prompt and response.

These similarity features explicitly encode alignment between components, providing direct signals for fitness prediction.

7.3 Weighted Cross-Validation

During model selection and hyperparameter tuning, cross-validation folds were weighted to reflect the score distribution. Folds were stratified by score levels, ensuring each fold maintained representative class frequencies. This prevented biased hyperparameter selection.

8 Result and Conclusion

The final ensemble model achieved a test RMSE of 2.290 (public) and 2.286 (private), with rank 30 out of 135 participants. The consistent performance across evaluation sets tells us that it did not overfit. Future work could explore deeper metric-specific architectures, advanced augmentation techniques like contrastive learning, and refined language-aware preprocessing.