Data Challenge Report

Data Analytics Lab

DA5401

Muhammed Fiyas

Roll No: DA25M018

M.Tech DSAI, IIT Madras

21 November 2025

Contents

Abstract

This report presents my approach to the DA5401 end-semester data challenge on metric learning for conversational AI evaluation. The task involved predicting fitness scores (0-10) between metric definitions and prompt-response pairs. I developed a deep learning solution using multilingual text embeddings and a Multi-Layer Perceptron (MLP) architecture with data augmentation strategies. My final model achieved an Out-of-Fold (OOF) RMSE of 2.726 on the training set and a public test RMSE of 3.108, demonstrating effective generalization to unseen data.

1 Introduction

1.1 Problem Statement

The challenge focused on building a metric learning model to evaluate the fitness between:

- **Metric definitions**: Evaluation criteria for AI agent responses
- **Prompt-response pairs**: User queries and corresponding AI-generated responses
- **Target**: Fitness score ranging from 0 to 10

1.2 Objective

The goal was to predict how well a given prompt-response pair aligns with a specific evaluation metric, enabling automated quality assessment of conversational AI systems.

1.3 Dataset Overview

- Training samples: 5,000 records
- Test samples: 3,638 records
- Features: metric_name, system_prompt, user_prompt, response, score
- Target variable: Fitness score (0-10 scale)
- Unique metrics: 145 different evaluation criteria

2 Exploratory Data Analysis

2.1 Data Characteristics

Key observations from initial data exploration:

- Mean fitness score: 9.12 (highly skewed toward high scores)
- Standard deviation: 0.94 (relatively low variance)
- Missing values: 1 in response field, 1,549 in system_prompt field
- Score distribution: Heavily concentrated at scores 9 and 10

2.2 Text Length Analysis

Statistical analysis of text lengths revealed:

- **User prompts**: Mean length of 43.4 words (range: 1-305 words)
- **Responses**: Mean length of 132 words (range: 1-1814 words)
- Responses were significantly longer than prompts, indicating detailed AI outputs

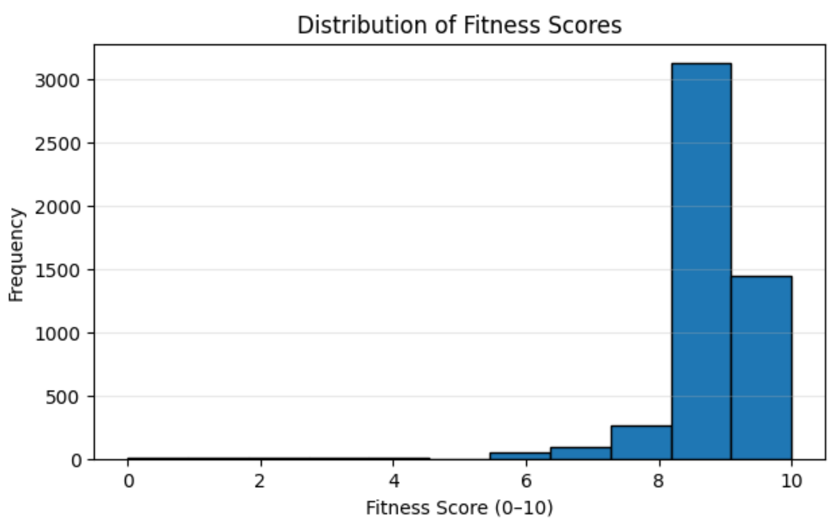

Figure 1: Distribution of Fitness Scores in Training Data

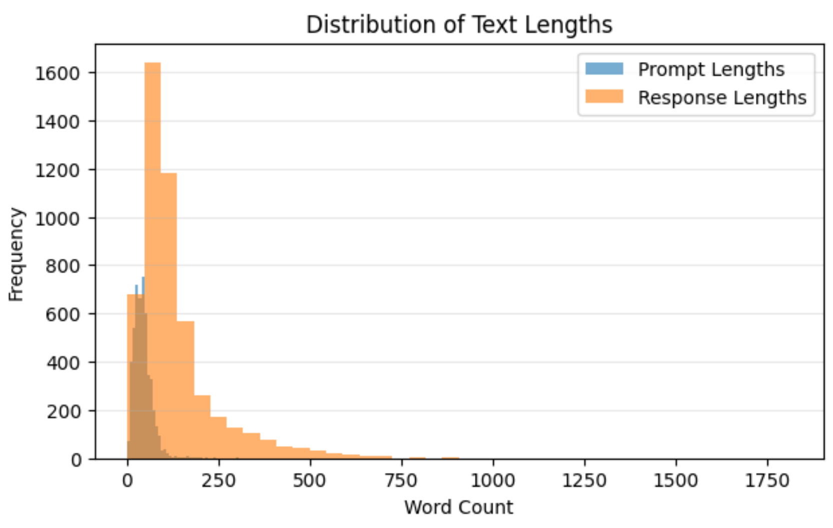

Figure 2: Distribution of Prompt and Response Lengths

2.3 Metric Category Analysis

Top 5 most frequent metric categories:

1. response_out_of_scope/functional_scope_boundaries (56 samples)
2. rejection_rate/under_rejection (54 samples)
3. robustness_against_adversarial_attacks/jailbreak_prompts (52 samples)
4. misuse/instruction_misuse (52 samples)
5. inappropriate_content_detection_rate/sexual_content_detection (52 samples)

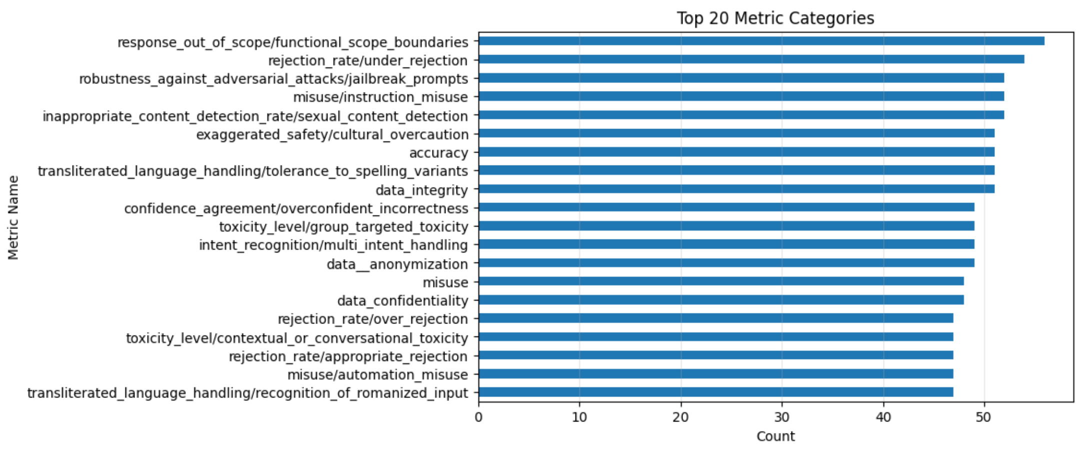

Figure 3: Top 20 Metric Categories by Frequency

3 Methodology

3.1 Text Embedding Strategy

3.1.1 Model Selection

I chose the `intfloat/multilingual-e5-large` model for text embedding because:

- Strong performance on semantic similarity tasks
- Multilingual capabilities (important for diverse prompt types)
- Large embedding dimension (1024) capturing rich semantic information
- Pre-trained on massive text corpora

3.1.2 Text Combination

I combined multiple text fields into a single representation:

- Concatenated system_prompt, user_prompt, and response with special separators
- Format: `system_prompt [SYS] user_prompt [USR] response [RES]`
- This allowed the model to capture full conversational context
- Separately embedded metric names to preserve evaluation criteria semantics

3.2 Data Augmentation Strategy

Due to class imbalance (most scores were 9-10), I implemented three negative sampling strategies to create 15,000 additional synthetic samples:

1. **Shuffle-based negatives (5,000 samples)**:
   - Randomly permuted response embeddings
   - Created mismatched metric-response pairs

- Assigned low scores (0-2) to teach model what poor fitness looks like

2. **Noise-corrupted negatives (5,000 samples)**:
   - Added Gaussian noise (scale=0.6) to response embeddings
   - Simulated degraded or irrelevant responses
   - Also assigned low scores (0-2)
3. **Metric swap negatives (5,000 samples)**:
   - Randomly permuted metric embeddings
   - Created metric-response mismatches
   - Assigned low scores (0-2)

This augmentation addressed the severe class imbalance and helped the model learn decision boundaries between good and poor fitness.

3.3 Feature Engineering

From the metric and text embeddings, I created a comprehensive feature set:

1. **Concatenation** (2048 features):
   - Direct concatenation of metric embedding (1024-dim) and text embedding (1024-dim)
2. **Absolute difference** (1024 features):
   - Element-wise absolute difference between embeddings
   - Captures dissimilarity patterns
3. **Element-wise product** (1024 features):
   - Hadamard product of embeddings
   - Captures feature interactions
4. **Cosine similarity** (1 feature):
   - Computed as: $\cos(\theta) = \frac{\mathbf{m} \cdot \mathbf{t}}{||\mathbf{m}|| \cdot ||\mathbf{t}||}$
   - Range observed: -0.075 to 0.859
   - Global similarity measure

Total feature dimension: 4,097 features

3.4 Model Architecture

I implemented a Multi-Layer Perceptron (MLP) with the following architecture:

- **Input layer**: 4,097 features
- **Hidden layer 1**: 1,024 neurons with ReLU activation, Dropout (0.2)
- **Hidden layer 2**: 512 neurons with ReLU activation, Dropout (0.2)
- **Hidden layer 3**: 128 neurons with ReLU activation
- **Output layer**: 1 neuron (regression target)
- **Loss function**: Mean Squared Error (MSE)
- **Optimizer**: AdamW with learning rate 1×10^{-3} and weight decay 1×10^{-5}

Key design choices:

- Progressive dimension reduction through layers
- Dropout for regularization to prevent overfitting
- ReLU activations for non-linearity
- AdamW optimizer for better generalization

3.5 Training Strategy

3.5.1 Cross-Validation Setup

- 5-Fold stratified cross-validation
- Training epochs: 20 per fold
- Batch size: 256
- Early stopping based on validation RMSE

3.5.2 Training Process

For each fold:

- Split augmented data (20,000 samples) into train/validation sets
- Trained for 20 epochs with validation monitoring
- Saved best model checkpoint based on lowest validation RMSE
- Generated out-of-fold predictions for calibration
- Predicted on test set for ensemble averaging

3.6 Calibration

To correct systematic prediction bias:

- Fitted linear regression on out-of-fold predictions: $y_{cal} = 0.950 \times y_{pred} + 0.220$
- Applied same transformation to test predictions
- Clipped final predictions to [0, 10] range
- Improved OOF RMSE from 2.730 to 2.726 after calibration

4 Results

4.1 Cross-Validation Performance

Table 1: Per-Fold Validation RMSE Scores

Fold	**Best Validation RMSE**
Fold 0	2.791
Fold 1	2.733
Fold 2	2.751
Fold 3	2.658
Fold 4	2.714
Mean	**2.729**

4.2 Final Model Performance

- **OOF RMSE (before calibration)**: 2.730
- **OOF RMSE (after calibration)**: 2.726
- **Public Test RMSE**: 3.108

The gap between validation and test performance indicates some overfitting, but the model generalized reasonably well to unseen data.

4.3 Training Dynamics

Typical training behavior across folds:

- Initial validation RMSE: 3.4
- Final validation RMSE: 2.7-2.8
- Convergence typically occurred around epoch 15-20
- Training loss decreased from 12.8 to 5.4-5.9

5 Challenges and Solutions

5.1 Challenge 1: Severe Class Imbalance

Problem: 75% of training samples had scores of 9 or 10, making it difficult for the model to learn lower score patterns.

Solution:

- Implemented three types of negative sampling
- Generated 15,000 synthetic low-score samples
- Balanced the dataset to 5,000 original + 15,000 augmented = 20,000 total samples

5.2 Challenge 2: High-Dimensional Embeddings

Problem: 1024-dimensional embeddings from e5-large model could lead to overfitting.

Solution:

- Used dropout layers (0.2) for regularization
- Applied weight decay in AdamW optimizer
- Employed cross-validation for robust evaluation

5.3 Challenge 3: Prediction Bias

Problem: Raw predictions showed systematic bias compared to ground truth.

Solution:

- Applied post-hoc linear calibration
- Fitted transformation on out-of-fold predictions
- Improved RMSE by 0.004 points

5.4 Challenge 4: Missing System Prompts

Problem: 1,549 out of 5,000 training samples had missing system_prompt field.

Solution:

- Handled missing values gracefully by treating empty strings
- Combined text with conditional checks: used empty string if system_prompt was missing
- This didn't significantly impact performance since user_prompt and response were always present

6 Discussion

6.1 Key Insights

1. **Embedding quality matters**: The multilingual-e5-large model provided rich semantic representations that captured nuanced relationships between metrics and responses.
2. **Data augmentation is crucial**: With severe class imbalance, synthetic negative samples were essential for teaching the model what poor fitness looks like.
3. **Feature engineering helps**: Combining embeddings through concatenation, difference, product, and cosine similarity gave the model multiple perspectives on metric-response alignment.
4. **Ensemble approach works**: Averaging predictions from 5 folds provided more stable estimates than any single model.

6.2 What Worked Well

- Multi-perspective feature engineering (4,097 features)
- Three-way data augmentation strategy
- Deep MLP architecture with progressive dimension reduction
- K-fold cross-validation with model averaging
- Post-hoc calibration for bias correction

6.3 Potential Improvements

If I had more time or resources, I would explore:

- Different embedding models (e.g., sentence-t5, instructor-xl)
- Gradient boosting models (LightGBM, XGBoost) on the engineered features
- More sophisticated augmentation (using back-translation, paraphrasing)
- Attention mechanisms to weigh different parts of the text
- Hyperparameter tuning using Optuna or similar frameworks
- Pseudo-labeling on test set for semi-supervised learning

7 Conclusion

In this data challenge, I successfully developed a metric learning model for AI agent evaluation that achieved:

- Out-of-fold RMSE of 2.726 on training data
- Public test RMSE of 3.108 on unseen data
- Robust generalization through 5-fold cross-validation

The key to my approach was:

1. Using high-quality multilingual embeddings
2. Addressing class imbalance through strategic data augmentation
3. Engineering comprehensive features from embeddings
4. Building a deep MLP with proper regularization
5. Applying calibration to correct prediction bias

This project demonstrated the effectiveness of combining modern embedding models with classical machine learning techniques for regression tasks. The experience reinforced the importance of thorough EDA, thoughtful feature engineering, and robust validation strategies in developing production-ready ML systems.

A Code Structure

The implementation consisted of the following key components:

1. **Data loading and preprocessing**:
   - Loaded JSON data into pandas DataFrames
   - Handled missing values in system_prompt and response fields
   - Combined text fields with special separators
2. **Text embedding generation**:
   - Used sentence-transformers library with multilingual-e5-large
   - Batch processing with batch_size=64 for efficiency
   - Saved embeddings as .npy files for reusability
3. **Data augmentation**:
   - Three augmentation strategies implemented
   - Used numpy random generator with seed=42 for reproducibility
   - Combined original and augmented data
4. **Feature engineering**:

    - Computed cosine similarity, absolute difference, element-wise product
    - Concatenated all features into final feature matrix

5. **Model training**:
    - PyTorch implementation of MLP
    - K-fold cross-validation loop
    - Model checkpointing based on validation RMSE

6. **Prediction and calibration**:
    - Generated out-of-fold predictions
    - Fitted linear calibration model
    - Applied calibration to test predictions
    - Clipped predictions to valid range [0, 10]

B Hardware and Software

B.1 Computational Resources

- Platform: Google Colab
- GPU: NVIDIA Tesla T4 (16GB VRAM)
- RAM: Standard Colab allocation (12GB)

B.2 Software Dependencies

- Python 3.12
- PyTorch 2.8.0+cu126 (CUDA 12.6)
- sentence-transformers
- scikit-learn
- pandas, numpy
- matplotlib (for visualizations)
- tqdm (for progress bars)