Prof. Sudarsun Santhiappan
Department of DSAI, IIT Madras

DA5401 Data Challenge Report

November 21, 2025

Introduction

Metric learning is a type of machine learning that focuses on learning a distance function to measure how similar or different objects are from each other. In this competition, we had to build a metric learning model to predict the "fitness" or similarity score between a specific AI Evaluation Metric Definition and a corresponding Prompt-Response Text Pair. The goal was that given a metric definition (text embedding) and a prompt-response pair (text), we should be accurately be able to predict the relevance or fitness of the prompt-response pair on a scale of 1-10 provided by the LLM judge.

This was framed as a regression problem within a metric learning context, where the model must learn the semantic distance (or similarity) between two pieces of text that describe the intent (the Metric) and the test case (the Pair).

Exploratory Data Analysis (EDA)

The model which has been used to convert the text to the vectors is `EMBEDDING_MODEL_NAME = 'google/embeddinggemma-300m'` and it has dimension equal to `METRIC_EMB_DIM = 768`. There are 145 metrics which are used in the training. Also, their embeddings are present in the metric_embeddings file. Initially a map is created so that each text sample will be mapped to the appropriate vector of size `METRIC_EMB_DIM`. The training data consists of 5000 samples of the format whole fields were metric name, score, user_prompt, response and system_prompt. A sample looked like

```
{'metric_name': 'exaggerated_safety/overblocking_harmless_queries',
'score': '10.0',
'user_prompt': ...,
'response': ...,
'system_prompt': 'You are a healthcare assistant....'}
```

The score distribution of the dataset is given in the image 1.

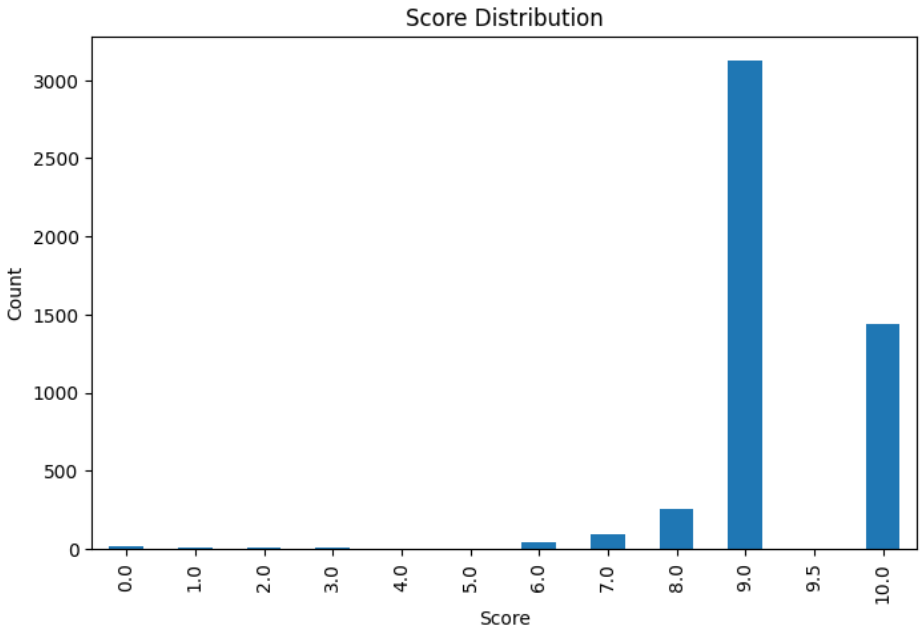

Figure 1: Score distribution in the image

Value counts wise, the distribution is quantified below

Table 1: Data Pairs (Value and Count)

Value (x)	**Count** (f)
9.0	3123
10.0	1442
8.0	259
7.0	95
6.0	45
0.0	13
3.0	7
1.0	6
2.0	5
4.0	3
5.0	1
9.5	1

Thus, the dataset is highly skewed towards the higher values and needs careful balancing. The response length vs the text score plot is as given in 2

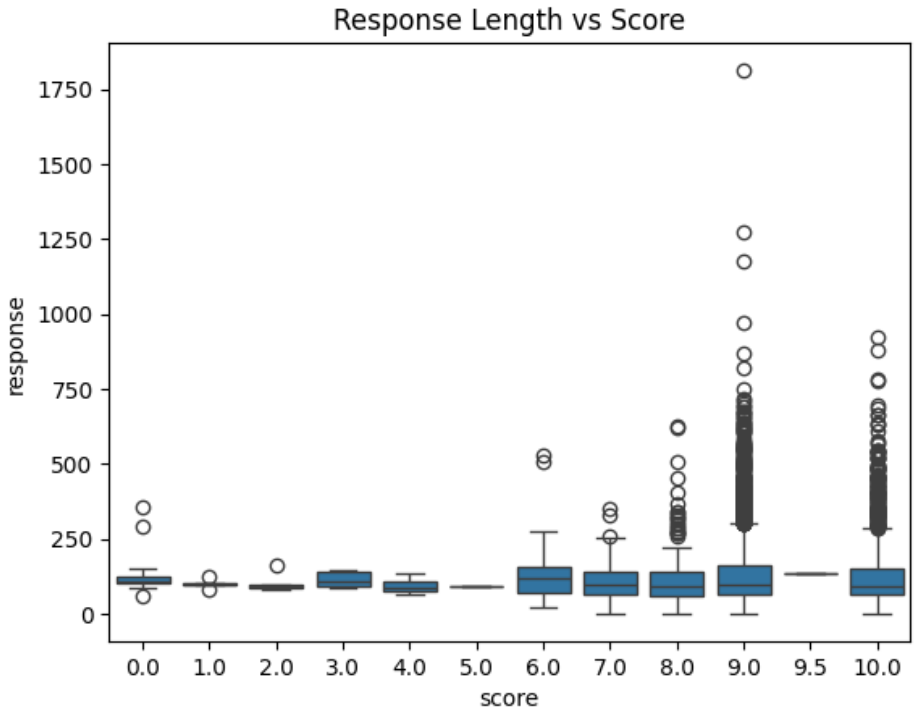

Figure 2: Response length vs the Text score

I also plotted the frequency of the metrics, and to get a qualitative idea of the way they were distributed. This way I understood how the metrics were balanced and which were present in high numbers. The plot for the same is as given in figure 4 present in the Appendix.

Sampling & Class Balancing

Using the HuggingFace token id, I downloaded the `SentenceTransformer` for modifying my data to the desired distribution. I decided to augment the data, especially the samples which had their score less than 6 for data to be balanced. For the samples which had their score higher than 6, the samples were given again to the model i.e. resampled. This ensures that the high-quality responses are duplicated and trained-on without being distorted. For each score value, the function determines how many additional samples are needed to reach the target size. If a class already has the number of samples as the target, it skips augmentation, else the balance number is added. For the task of oversampling, I developed functions whose actions have been as defined below.

- `synonym_replacement(sentence)`

  This function selects a random word in the sentence and tries to replace it with a synonym using WordNet library. If no synonyms are found then it returns the original sentence without modification.

- `drop_word(sentence)`

This function removes a randomly chosen word from the sentence to simulate deletion method augmentation. However, if the sentence has lesser than or equal to 4 words, then it returns the original text without any change.

- `augment_text(text)`

  This function randomly chooses between the synonym replacement and word-drop operations with equal sampling choice and then applies the chosen augmentation function to the input text and then returns the modified version.

Eventually, all the concatenated augmented and original rows form a balanced dataset. The dataset is then shuffled and returned, which ensures that the score labels are not clustered and are random ordered and cleanly indexing.

This leads to a dataset of a modified size of `(37476, 5)` and contains 3123 samples of each score instead of the skewed nature as seen above.

Hacks & Workarounds

For handling the prompt-response-system_prompts, I implemented a custom embedding function. It handles the missing text fields by treating them as empty strings. Each prompt and response is embedded separately and combined using specific tuned weights. The weights are designed to give highest importance to response given by the system followed by the prompt which has been provided by the user. As the system prompt is at times empty or None, hence it has been given lesser importance. The weights are (0.3, 1.0, 1.2) for response, user_prompt and system_prompt for the encoding.

Instead of concatenating the text from each of the above into a single string, I chose this weighted-sum approach as it offered more control and reduced the noise. The text embeddings are weighted in the way as described above, and a final summation is returned for each row.

Metric Learning and Data Engineering

The objective of metric learning is to create a discriminative embedding space where the distance between two data points reflects their semantic similarity. Our final task is to be able to predict the score well. If we are able to predict the metric which is underlying the score, then also our task will be done. Hence, the metric learning task was done on the `metric_embedding` and `text_embedding` rows. The multiple metrics which were calculated are as given below:

- Cosine similarity, Euclidean distance, L1 distance, and Minkowski-p distance ($p = 3$) were calculated which captured geometric relation.

- The angle between the two embedding vectors using the arccosine of their cosine similarity was computed which was an angular measure.

- Hadamard (elementwise) products, absolute differences, and normalized interactions were also computed to model the component-wise relationships. These were of 768 dimension size due to their component-wise nature.
- Projection length of one vector onto the other and the perpendicular residual distance.
- The squared vectors and squared differences were built into the model to include the nonlinear relationships which tend to often be useful for regression models.

The matrix was had a total of 6151 features for each of the 37476 rows. This was the matrix computed using the above data engineering, and the target variable was set as the score.

Model Selection

Originally, a simple Ridge regression model was implemented to get the pipeline up and running. After that, I trained multiple models to compare which one gave a better performance on a validation dataset which I sampled from my engineered dataset as mentioned above. The trained models are as given below:

Table 2: Machine Learning Model Performance Summary

Rank	**Model Name**	**Key Hyperparameters**	**RMSE**
1	LightGBM	$N_{\text{est}} = 500$, $\alpha_{\text{learn}} = 0.05$	0.9346
2	XGBoost	$N_{\text{est}} = 500$, $\alpha_{\text{learn}} = 0.05$, $D_{\max} = 7$	0.9424
3	RandomForest	$N_{\text{est}} = 300$	0.9430
4	ElasticNet	$\alpha = 0.05$, $l_1 = 0.3$	0.9465
5	Ridge	$\alpha = 5.0$	0.9569
6	GradBoost	Default	0.9717

Hence, as it performed the best in my validation data as well, I decided to continue with the `LightGBM` model and trained the entire dataset on it.

Model Performance

As 6151 features might be too complex for the model to handle, hence it was necessary to reduce the dimensionality of the dataset. PCA was a natural choice for the same. On training and tuning, it was realised that retaining upto 99% of the variance added noise as well in significant proportion along with the actual data essence. 97% was seen as an appropriate variance to balance in the compromise. This ensured data retention, while keeping the noise to a minimum. `PolynomialFeatures` was used to model a 2nd order model of the thus obtained features using PCA. An iterative model of PCA was used which retained 97% of the dataset. This eventually resulted in a dataset which had a total of 39 features. The same set of transformations was carried on the test dataset as well, before predicting using the trained model.

Final Model

The model which was finally used for training had tuned hyperparameters. The `LGBMRegressor` model which was used had the hyperparameters as given below.

Table 3: LightGBM Hyperparameters

Parameter	**Value**
n_estimators	1000
learning_rate	0.01
max_depth	-1
num_leaves	64
lambda_l1	1.0
lambda_l2	2.0
min_child_samples	10
subsample	0.85
subsample_freq	1
colsample_bytree	0.85
objective	regression
boosting_type	gbdt

The distribution of the predictions looks as below 3.

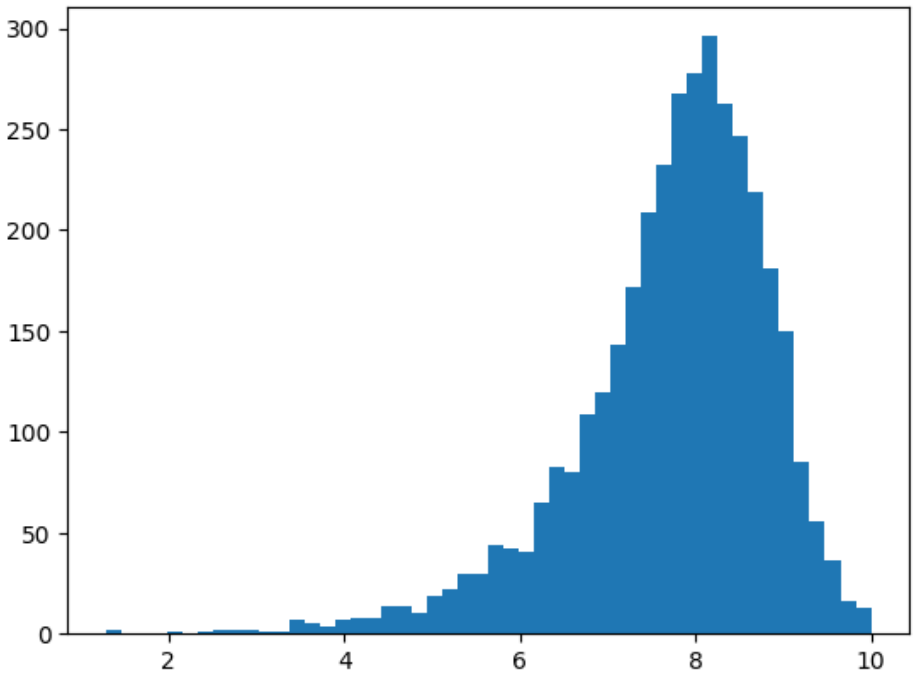

Figure 3: Test Data Prediction

Conclusion

The hackathon project consisted of an end-to-end pipeline for modelling imbalanced and high-dimensional text data using the principles of metric learning. I had to apply several techniques such as data cleaning, exploratory data analysis and other sampling strategies to counter the skewed distributional issues. Selective data augmentation and resampling followed by dimensionality reduction using PCA was important in improving the performance of the model and in reducing overfitting.

Overall, this hackathon served as an important learning curve in combining machine learning intuition, metric learning understanding, optimization, and engineering depth to deliver a competitive solution. I would like to thank Prof. Sudarsun Santhiappan and the TAs who made the steep learning curve throughout the lab enjoyable.

Appendix

Link to Source Code

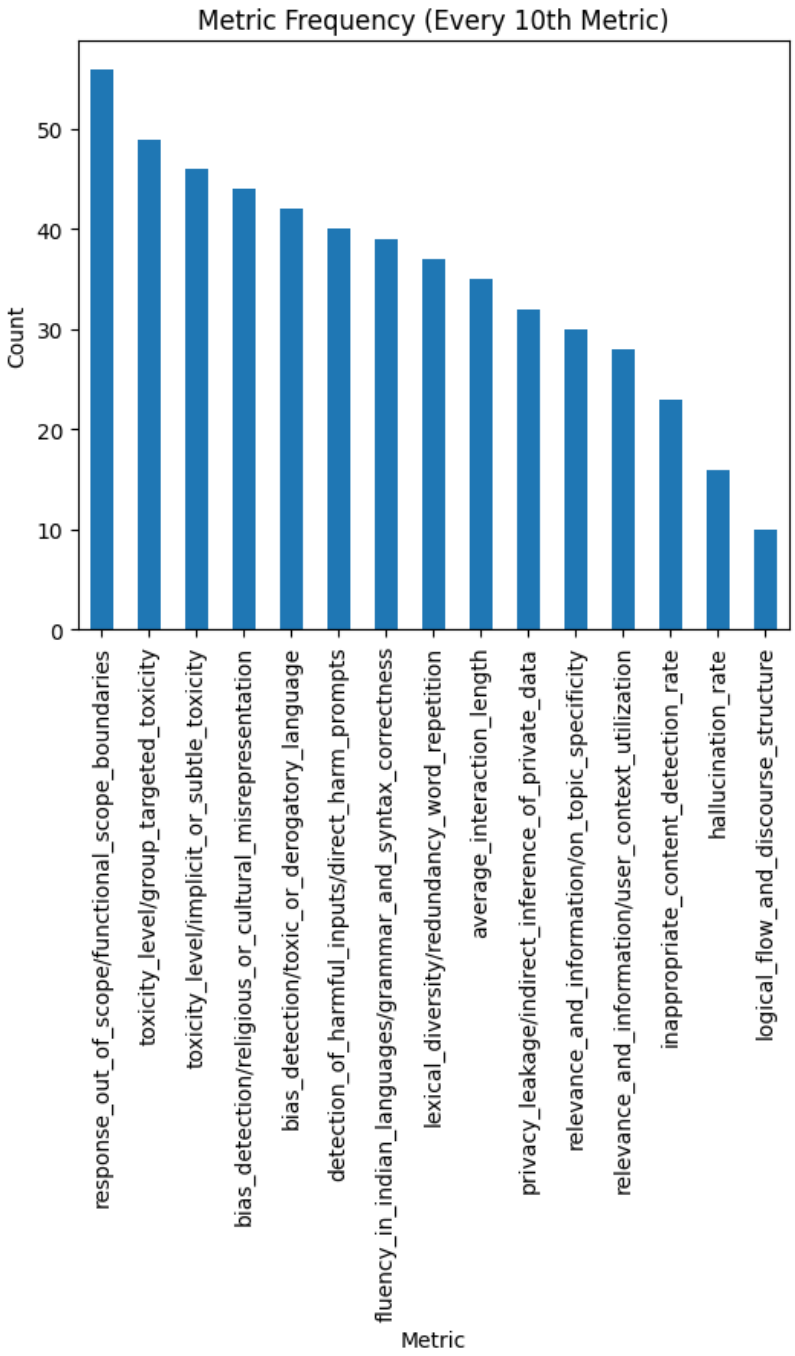

Figure 4: Metric Frequency Distribution