Kaggle Competition Report

Problem Formulation

The goal of the competition is to predict relevance or fitness on a scale of 1-10 given a metric definition (text embedding) and a prompt-response pair (text).

I have formulated this problem as a metric learning problem. The core idea implemented is metric learning (inspired by that book attached, "Metric Learning" by Bellet et al.). Metric learning learns a "distance" or "similarity" space where similar items cluster close together, and dissimilar ones are far apart. The loss function's job is; given an anchor (a data point), a positive (similar score), and a negative (different score), force the model to make anchor-positive distance smaller than anchor-negative by a margin. Simply put, the loss function tries to cluster similar points together while distancing them from dissimilar ones in a latent space. The main model acts as an embedding refiner. It takes the raw combined embeddings and outputs a new, tuned representation space.

The output is a matrix of shape (n_datapoints, embedding_dimension); ex. For 5000 train datapoints, the matrix is of shape (5000, 768). Each row is a refined embedding for one data point.

However, that is not the end of the story. To infer the "score" as required by the competition, a much simpler second model is trained. The job of this model is to convert the (5000, 768) matrix into 5000 predictions of score. (In implementation, another small step is added)

Data Engineering

Data preparation

The dataset given for this competition is in JSONs. Embeddings for the metric definitions are given to supplement. The main steps taken in data engineering are:

1. Extracting data from given JSONs
2. Generating Gemma embeddings from system prompt, user prompt, and response
3. Mapping metric names to metric name embeddings
4. Creating an input matrix by stacking the 4 input items:
    a. Metric name embedding - mapping
    b. System prompt embedding - generated
    c. User prompt embedding - generated
    d. Response embedding - generated
5. Extracting score values from JSONs

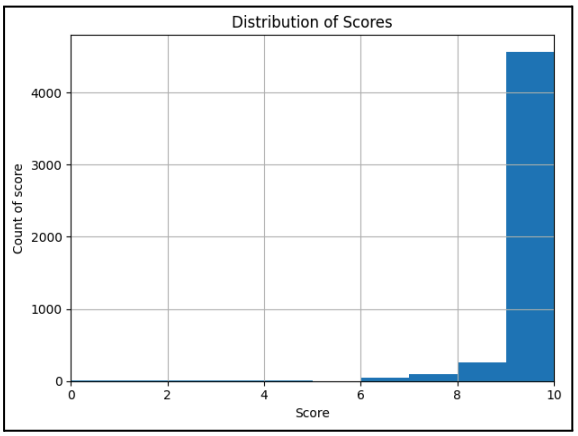
The resultant input matrix is of shape (embedding shape, 4 total embeddings, n datapoints). This results in a size of (768, 4, 5000) for the training matrix and (768, 4, 3638) for the testing matrix. The score vector is just a list of 5000 values for the train set.

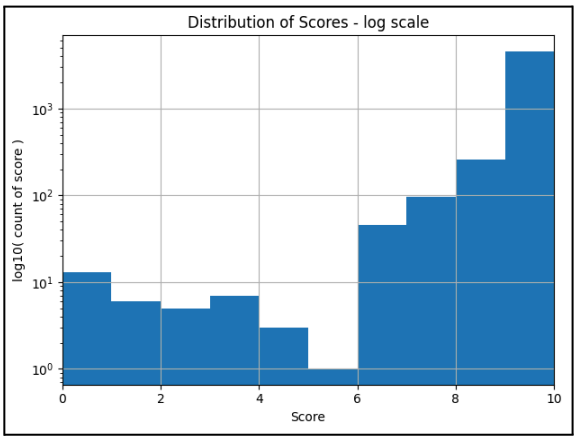
Atharva Waghmare
DA25C004

Kaggle Competition Report

Visualisation and EDA

The data from the JSONs and the matrix has been sampled and printed to visualise. EDA revealed that the input data is highly skewed toward higher values of score.





As the 2 plots show, the number of data-points for scores over 8 far out-weight the number of datapoints below 8 (especially 5 and below).

This seems like class imbalance but after experimenting with the test data (via submissions to Kaggle), I discovered that the test distribution is the complete opposite of the train. This is NOT class imbalance, it is a complete disconnect between train and test. The solution to this however, was not a part of data engineering. It is explained further in the implementation section.

Model Architecture

The solution uses 2 separately trained models:

1. **Model 1:** Embedding refiner

This model is an attention based model. It takes the input matrix, splits it into 2 matrices; one includes the metric name embedding while the other has system prompt, user prompt, and response embeddings. These 2 matrices are then passed through a series of multi-head self attention layers and the output is passed into a series of cross attention layers which then feed into the last set of multi-head self attention layers that output the distance matrix.

This model is trained using the triplet hinge loss mentioned in the Metric Learning book. The formula is given below:
Loss = max(0, distance_to_similar – distance_to_dissimilar + margin)

The idea of having this model is to represent the inputs in a latent space where they are better clustered.

2. **Model 2:** Regressor

This model is a Ridge Regressor. It takes the output of model 1 and maps it to the score.

Atharva Waghmare
DA25C004

Kaggle Competition Report

Implementation

The final implementation of this idea was in the form of a few architecture classes, one model class, one loss function class and the functions for train, eval and test.

Loss function

Firstly, the loss function was implemented as per the book. The TripletHingeLoss class uses the concepts in the book and outputs a loss value. It takes into account the margin as well. Other loss functions were tried but this was the best of the bunch

Cross Attention

Cross attention is a powerful technique used to relate between 2 different inputs. Each input being the output of multiple MHSA (multi-head self attention) layers from either the metric name embedding or the prompts and response embedding. This is one of the most central modules of the architecture. The cross attention mechanism maps the prompts and the response to the metric.

Model training and evaluation

The functions for model training and evaluation along with the setup for model training are implemented. These are more or less standard functions. The model is trained on Kaggle using GPUs.

Regression model

The second model, used for mapping the distance matrix to the score, is a simple ridge regression model.

Output scaling

As seen in previous sections, the data given is not just skewed, it is completely different from the test data. Under normal conditions, this would be considered a class imbalance issue. However, since the amount of data is also not massive, the class imbalance turns into a dichotomy of picking the data for training or validation.

Due to the lack of data for lower values of score, we end up with a choice with no correct answer; either we take whatever little data there is and use it all to train the model and end up with an irrelevant validation set or we keep some data for validation and the model doesn't have enough data to train on.

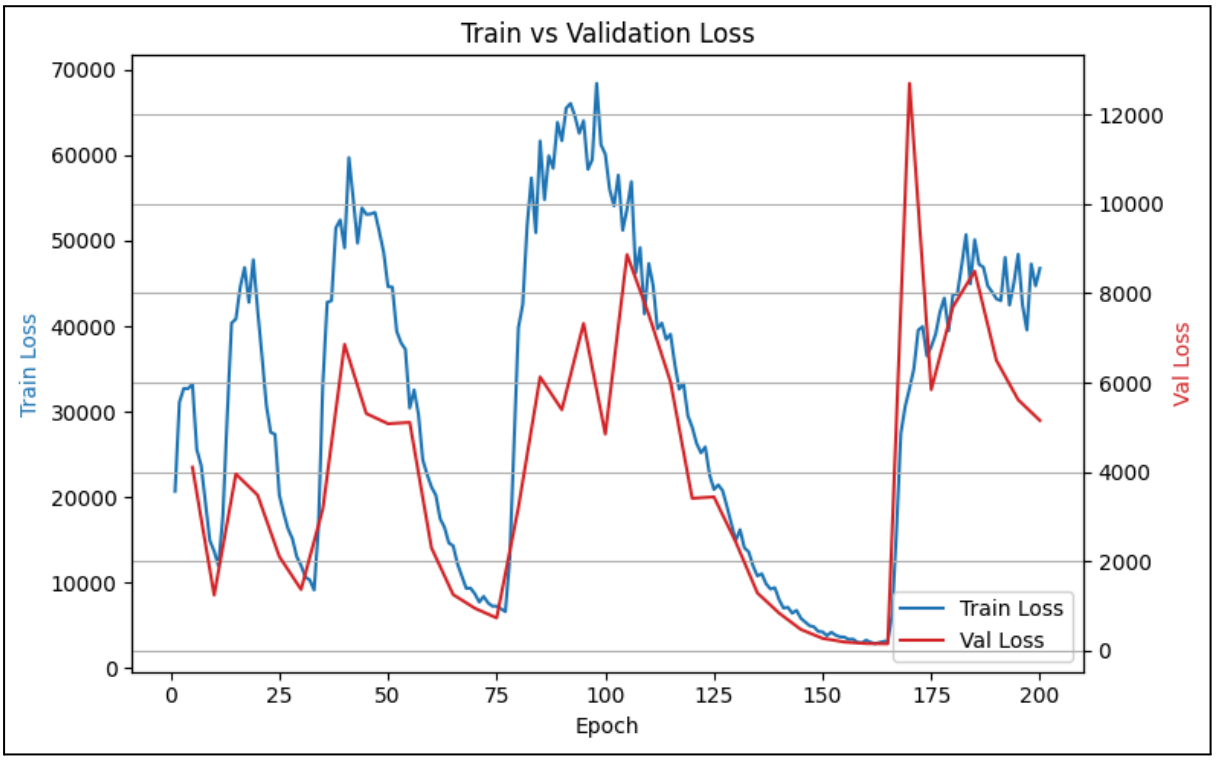
The solution that I implemented for this problem was to assume that, even though the scores in the train data and test data have vastly different means, they have similar underlying distributions. This is akin to assuming the variance of the 2 datasets is the same and testing for the mean in hypothesis testing. In the implementation, I have used this idea and applied it to the test outputs. The predictions from the Ridge model for scores was shifted by 3 to match the assumed mean of the test distribution.

Atharva Waghmare
DA25C004

Kaggle Competition Report

Loss Visualisation

Training and validation performance tables of the final model.



Atharva Waghmare
DA25C004