DA5401: Data Challenge Report

Siddharth M
DA25C021

November 21, 2025

1 Exploratory Data Analysis and Challenges Faced

The Kaggle Data Challenge was an enlightening experience as it simulated a real world environment with poor quality data. There were two main challenges that I faced while analyzing the datasets.

1. Feature extraction from the high dimensional embeddings
2. Severe skewness of the dataset, with most points concentrated around the 9-10 range.

I had taken several approaches to remedy these, which I will discuss below.

1.1 Feature Extraction

I began by visualizing the similarity metrics with respect to the scores, as seen in Figure 1.1. The standard similarity metrics showed poor correlations with the scores and were not separable by any means in the vector space. Thus, I realized that different features are required that are non standard to address this concern. I had also visualized the embeddings directly in two and three dimensions using TSNE to see if the embeddings themselves were sufficient to separate out the dataset. However, this too resulted in a very mixed and non-separable data.

1.2 Class Imbalance

The other glaring problem with the dataset was the imbalance or skewness in the scores. As seen in Figure **??**, the majority of the scores are on the higher end, with very few on the lower side. This means that the model will learn to predict the majority score each time and not be penalized by much through misclassifications of the minority scores. I had taken the following approaches to deal with this situation.
After further scrutiny, I had decided to let the model extract features by feeding it both the prompt-response embeddings and the metric embeddings alongside the distance metrics. I had ruled out SMOTE due to the substantial scarcity of data in the lower scores, as this would cause the generation of noisy samples.

1. Used tree based boosting algorithms as they are robust against class imbalance and outliers in the dataset.

2. Oversampled the minority classes using random sampling or SMOTE.
3. Undersampled the majority classes through random sampling.
4. Used a weighted loss function to punish the model more for incorrectly scoring the minority classes.

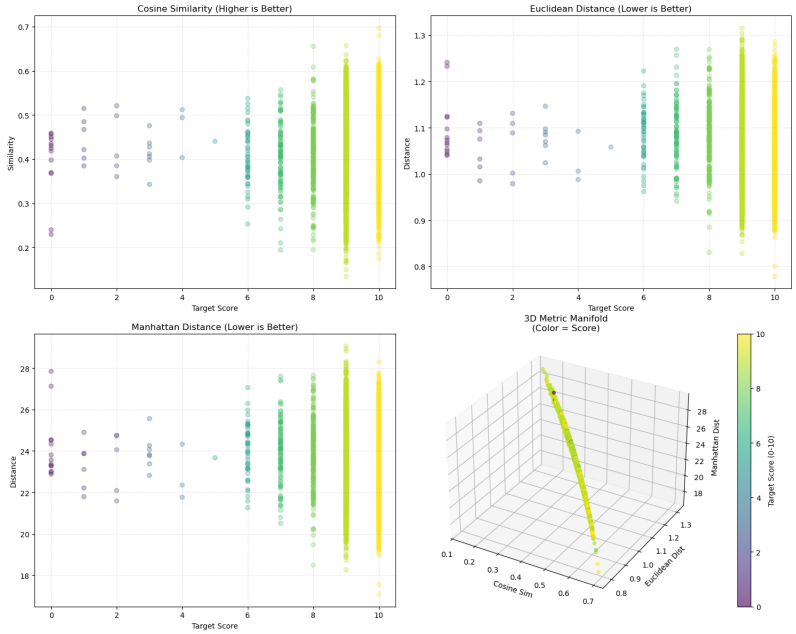
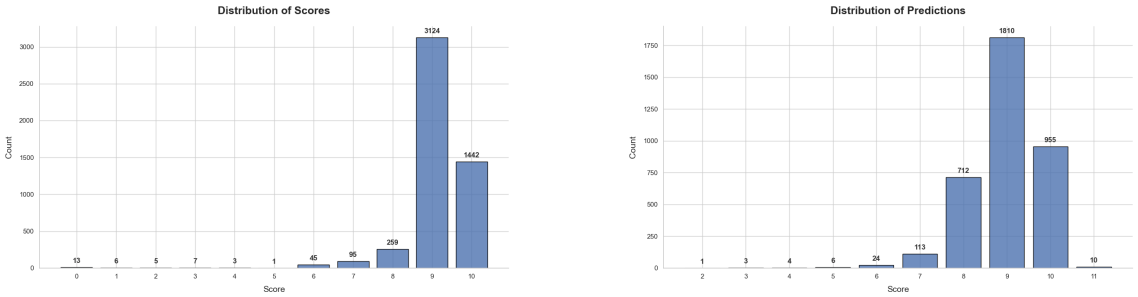


Figure 1.1: Distance Metrics Visualizations



(a) Dataset

(b) Best model predictions

Figure 1.2: Distribution of scores

1.3 Data Augmentation

A dataset consisting of text is grounds for several data augmentation practices frequently used in NLP. I had considered this approach as well to generate negative samples, but faced several roadblocks, which are:

1. The majority of the minority samples were written in indic language and thus techniques such as *synonym replacement* and *random insertions* are not possible due to the lack of good language models for these tasks.
2. *Back translation* was also not feasible for the same reason.
3. *Sentence Shuffling* might have worked but since the prompt-response pairs had very few sentences, this wouldn't have resulted in a sufficient increase. I was also worried about

breaking the grammar and logical flow of the texts which could potentially impact the model negatively.

I had considered dealing with the embeddings directly by applying linear transformations to generate negative samples (such as multiplying by -1). However, the embeddings space wasn't a good representative for the semantics and hence I decided to drop this idea as well.

2 Model Selection

Initially, I had intended on using gradient boosted tree based models such as *XGBoost* and *LightGBM*. I was quite optimistic about them as they are robust against class imbalance, which seemed to be the main problem. However, they ended up overfitting the training data repeatedly and showed poor test data scores. The performance marginally improved when I took a cost-sensitive approach by providing sample weights based on the proportion of data of a particular score.

Next, I moved on to *Support Vector Regressors* hoping for a different result. I realized that normalizing the embeddings would make them sit in a hypersphere, and thus a kernel based SVR might be able to learn from the data. I applied randomized oversampling to the embeddings of the minority classes and creating a ten-fold followed by a fifty-fold increase in their samples sizes and performed hyperparameter tuning on the *RBF-Kernelized SVR*. This model performed worse than the gradient boosted trees and was also computationally expensive to run.

I was also looking at metric learning models such as the *Information Theoretic Metric Learning* to perform this task as it seemed to be quite relevant. However, I was hesitant to use it as it requires categorizing points into similar and dissimilar categories which would result in the loss of information such as the score assigned. However, I did utilize the *Siamese-Neural-Network* (Figure 2.1), a deep learning model with a similar usecase. I picked this as I wanted to change the loss function from the contrastive loss function to mean squared error in order to use it to perform regression. I augmented the training of this model with both *Random Oversampling* of the minority classes with a shrinkage parameter and *Random Undersampling* with the majority class in order to address the class imbalance. I also created a custom weighted mean-squared error loss function to further address the class imbalance issue.

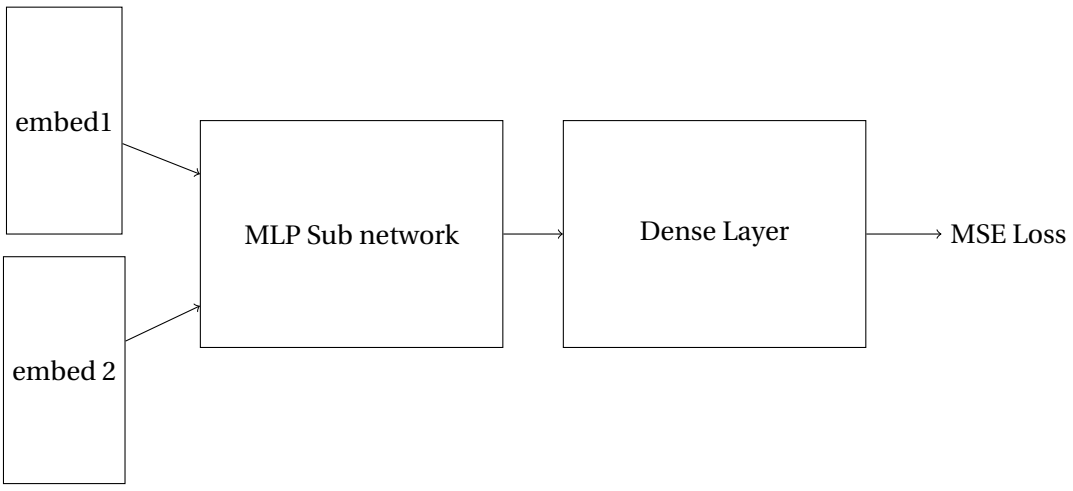


Figure 2.1: Siamese-style neural network for contrastive learning with modified loss function

3 Model Performance

The *Siamese Neural Network* performed worse than expected likely due to the unfit loss function which prevented it from doing any meaningful metric learning. The model consistently overfit the train data while also not learning to categorize the minority classes despite the weighting and random sampling. Changing the activation functions from *ReLU* to *Tanh* did give a minor performance boost but it nevertheless was biased towards predicting the majority classes. I simplified the model by reducing the number of layers and also tried to truncate the embeddings due to its nature of being *Matryoshka Embeddings* to reduce the number of model parameters but this too did not result in an improvement.

What did help was to oversample all the minority classes to match that of the majority classes after a reasonable level of undersampling even though this would produce substantially noisy data. This gave me the best result in the test set (Figure 1.2b), beating the previous score set by the *Weighted LightGBM Model*.

What surprised me was that the validation loss was higher for this set of parameters than it was for the rest. The validation loss was around 2.05 while the other configurations of this model resulted in losses nearing 1. A reason for this could be that the validation set is also as skewed as the dataset which causes the biased models to show better performance as they would predict the majority samples more than they would predict the minority samples. In hindsight, I should have used the *Weighted MSE* loss function for evaluating the validation set as well as opposed to the standard *MSE* loss function to provide a more accurate performance metric. The model performance is summarized in Table 3.1.

Train Loss (WMSE)	0.57
Validation Loss (MSE)	2.05

Table 3.1: Training and Validation Loss for the best model.

4 Conclusion

Overall, none of the models could satisfactorily adapt to the imbalance in the dataset and ended up performing poorly in the competition. The models could be improved by primarily learning a good distance function to separate the data based on scores. Following this, a strategy to tackle class imbalance is also required. For the former task, we can expect the Siamese-Neural-Network to perform well at learning a good distance function using the contrastive loss, following which, gradient boosted tree based models could be used to learn the relationship between the features and the scores.