Metric learning and fitness of LLM response

PUNEET MISHRA

DA25C016

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Abstract

LLM apps like ChatGPT, Perplexity and Gemini have become an integral part of people's life. OpenAI's ChatGPT receives over **2.5 billion prompts** from users worldwide every day and other models have similar demand as well. It is crucial for these models to give appropriate and desirable responses to these prompts to provide their customers the best service possible. The machines must be trained on enormous data of prompt-response pairs and their fitness score in order to learn the difference between an unfit and a proper response. Here generating fitness scores that are close to human fitness score itself is as important task as making the machine learn the pattern. It is truly a herculean task and worldwide hundreds of researchers are working in order to formulate the best scoring method. *Metric Learning* is one of the most effective methods out there. As a part of the DA5401 course, I implemented one of most widely used Metric Learning algorithms to build a robust model that generates fitness scores for multilingual prompt-response. This report details the implementation of the algorithms as well as the results that were obtained.

Keywords: Metric learning, similarity learning, fitness score

1. Introduction

But what is "Metric"?

It is imperative to set the definitions of "Metric" and "Metric Learning" clear before presenting their implementation and performance.

A blunt, simple definition of Metric is – *"a pairwise function that returns the distance or similarity between two points in feature space".* A more detailed definition would be –

A metric is a distance function that satisfies the following four conditions (Bellet, Habrard, & Sebban, 2015) –

1. $d(x, x') \geq 0$ (nonnegativity),

2. $d(x, x') = 0$ iff $x = x'$ (identity of indiscernibles),

3. $d(x, x') = d(x', x)$ (symmetry),

4. $d(x, x'') \leq d(x, x') + d(x', x'')$ (triangle inequality).

There are several distance functions that are used in metric learning, the most prominent being *Euclidean distance, Mahalanobis distance* and *cosine similarity.*

Euclidean distance is simply the distance between two points in the space. The cosine similarity is the cosine of the angle between them. Mahalanobis distance is defined as $d_M(x, x') = \sqrt{(x - x')^T M (x - x')}$ where M is the parameter matrix. This metric is used in this metric learning project to learn the pattern between prompt-response pair and criteria definitions to generate fitness score.

But what is "Metric Learning"?

The goal of metric learning is to adapt some pairwise real-valued *metric*, say the Mahalanobis distance to the problem of interest using the information brought by training examples. (Bellet, Habrard, & Sebban, 2015). A metric learning algorithm basically aims at finding the parameters of the metric (here, the matrix ***M***) such that it best agrees with these constraints, in an effort to approximate the underlying semantic metric.

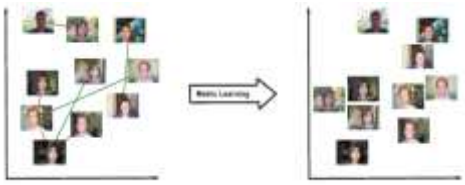


Figure 1.1: Illustration of metric learning applied to a face recognition task. For simplicity, images are represented as points in two dimensions. Pairwise constraints, shown in the left pane, are composed of images representing the same person (must-link, shown in green) or different persons (cannot-link, shown in red). We wish to adapt the metric so that there are fewer constraint violations (right pane). Images are taken from the Caltech Faces dataset.[1]

(Image from *Bellet et. al.*)

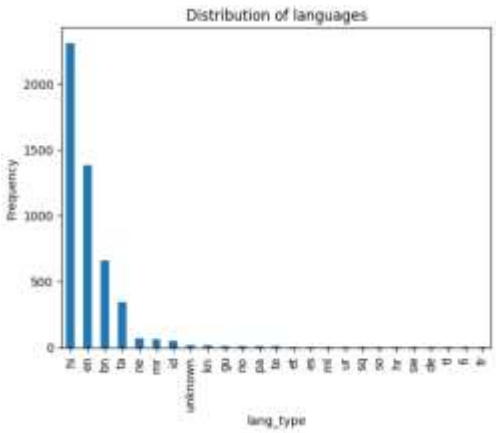
What is the aim here?

Metric learning was applied in this project to learn how well a conversational AI agent generates response to a prompt while bounded by a "metric" (the metric here is a criterion of sort on which the response is evaluated). The goal was to learn the parameters of Mahalanobis matrix M to predict the fitness score of an unseen prompt-response pair.

2. The Data

Train Data

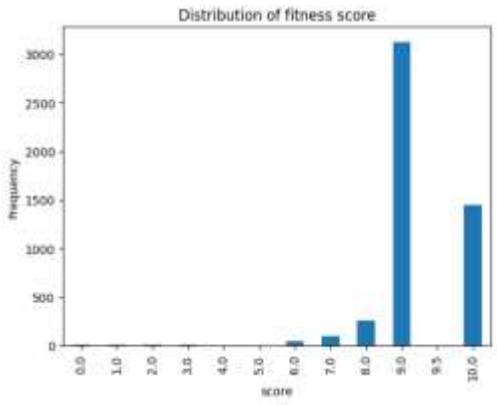
The training data was comprised of four features – *metric_name* like *"rejection rate"* or *"bias detection"* and other similar criteria on which the response was evaluated (metric names were given separately packed in **metric_names.json)**, *user_prompt* which was the prompt given by a human user, *system_prompt* is the prompt generated by the inner mechanics of the agent itself to guide its response, *response* i.e. the response generated by the model – and the *"score"* target variable which is the fitness score that an LLM judge has assigned to the prompt-response (P-R) pair in the range of 0-10. This data was packed into **train.json** file. The prompt-response pairs were multilingual comprising of a band of languages. The *system_prompt* on the other hand was mainly in English.



(the meaning of abbreviations is given in the notebook)

The distribution of fitness score was discovered to be highly skewed toward the high scores.

This made the training of the ML model particularly challenging as overfitting had to be avoided.



metric_name_embeddings.npy

This numpy file had the *"embeddings"* of the definitions of the metric names (Embeddings refer to the vectorized representation of text in some high-dimension space).

Test Data

The test data, packed in **test.json** had similar features like the train data, except the *"score"* column, naturally. The task was to train the model on the given train data to generate fitness score predictions for P-R pairs in test data. The predictions were evaluated by examiners using the ground truth data and scores were given to each set of predictions based on the *Root Mean Squared Error (RMSE)* between the predictions and actual score.

3. Methodology

Data pre-processing

It is the standard in machine learning practice to clean the data before passing it to the model. The train data has multiple anomalies that had to be removed.

a) *score* and *P-R pairs* had certain outliers, such as an anomalous score of 9.5 and some non-text elements in P-R pairs.
b) *system_prompt* had multiple redundant words such as 'you', 'are', 'a', 'who', etc. Though these words are important when it comes to the verbal syntax of the string, they are of no great use in machine learning. If anything, these add noise in embeddings. The embeddings generated after removing such "stop words" are more direct and useful.
c) The data had to be split into "train" and "validation" sets to evaluate whether the model chosen for learning is right.

Once these cleaning and preprocessing steps were completed, the data was ready to be fed to the ML model.

The Metric Learning Model

The model chosen to learn the metrics was a non-linear model which is fit to learn multiple local metrics. Why? The given data is extremely diverse. Not only it is multilingual, it is also multi-metric. If instead any linear ML model were chosen, it would have given faulty results due to its high bias of linearity. A non-linear model therefore was more suitable to learn different local metrics.

The model chosen was *Sparse Compositional Metric Learning (SCML).* SCML learns a data-dependent metric by expressing each local metric as a sparse, nonnegative combination of a shared set of rank-one basis matrices. Instead of fixing one global metric, SCML assigns each point a weight vector over these bases through a smooth mapping , which defines the local distance

$$\mathrm{d_W}(x - x') = (x - x')^T \left(\sum_{i=1}^{K} W(x)_i b_i b_i^T \right) (x - x')$$

The weights are parameterized as , $[W(x)_i = (a_i^T P(x) + c_i)^2$ ensuring nonnegativity and allowing nonlinear variation across the space. A kernel PCA projection reduces dimensionality and controls complexity. Learning minimizes a hinge-loss objective over triplets that encourages correct relative distances while applying a mixed norm to enforce sparsity across bases. This yields a compact, locally adaptive metric that shares structure across the dataset and captures manifold geometry while avoiding overfitting.

The Implementation

The `SCMLRegressor` pipeline implements an SCML-style regressor over concatenated metric-name embeddings and transformer-encoded text features. The functions as given in the source notebook perform the following functions –

`SCMLRegressor`
Defines a differentiable SCML block. It has the following elements –

- Input ($x \in \mathbb{R}^d$).
- A fixed basis matrix ($B \in \mathbb{R}^{\{d \times K\}}$ is stored as a non-trainable buffer.
- A linear projection P(x) is applied via `self.proj`.
- Weights $w(x)$ are produced by a linear layer followed by softplus to enforce non-negativity.
- Basis scores ($s = x^\top B$).
- SCML features ($\phi_i(x) = \sqrt{\{w_i(x)\}}\, s_i$).
- A small MLP maps $\phi(x)$ to a scalar regression output.

`TensorDatasetReg`
Wraps X and y into a PyTorch dataset.

`SCMLPipeline` initialisation

- Loads transformer encoder (tokenizer + model).
- Loads metric-name embeddings and builds a lookup map.
- Initialises PCA (not fitted yet).
- Model is set to None until training.

`_load_df`
Reads JSON into a DataFrame.

`_align_metric_emb`
Maps each metric_name to its pre-computed embedding, producing an aligned matrix.

`_encode_texts`
Encodes concatenated [user_prompt | system_prompt | response] using the transformer. Tokenizes in batches, runs the model in eval mode, extracts mean-pooled hidden states, returns embeddings.

`_build_features`
Concatenates metric-name embeddings and transformer text embeddings into a single feature matrix.

`train`

- Loads training JSON.
- Builds raw feature matrix [metric | text].

- Fits PCA to reduce dimensionality.
- Creates SCML basis: uses identity basis (each basis vector is a standard coordinate vector), producing (K) rank-1 directions.
- Splits into train/validation. Initialises `SCMLRegressor` with given basis.
- Trains with `AdamW` (a type of optimizer) and MSE loss.
- Tracks validation RMSE.
- Applies early stopping. Stores best model state and PCA-transformed training data.

`predict_test`

- Builds test features in the same way, applies fitted PCA.
- Runs the trained `SCMLRegressor` to produce predictions.
- Rounds and clips values to [0, 10] and produces results in .csv format.

4. Results

The SCML model gives highly accurate results. The RMSE obtained by predictions on test data was consistently between 3 and 4 thus the model gave precise and near-accurate results. With some hyperparameter tuning, the RMSE dropped to as low as **2.889.** However, it was observed that even after heavy tuning, the RMSE remained saturated between 3 and 4.

Why SCML gave saturated RMSE?

SCML saturates at RMSE ~3–4 for three structural reasons:

1. **Ineffective metric basis.**
   The implementation uses identity vectors as basis elements, so the "metric" reduces to per-coordinate gating rather than learning meaningful rank-one components. The SCML block cannot model local geometry or compositional metrics, causing persistent underfitting.
2. **Weak downstream capacity on frozen embeddings.**
   The transformer encoder is fixed, and the regression head is shallow (linear projection → softplus weights → small MLP). With complex, noisy text–metric relationships, this architecture lacks representational power, driving predictions toward mid-range values and capping RMSE.
3. **Information loss and over-regularisation.**
   PCA removes low-variance but possibly predictive directions; dropout and weight decay further suppress signal; early stopping halts training before deeper fitting occurs. The compound effect strips essential structure from the feature space, locking performance into the 3–4 RMSE band.

5. Conclusion

This study illustrates how metric learning can be applied to multilingual P-R evaluation. The model is simple (relative to large LLM judge models used by chatbots), mathematically grounded and largely explainable. It uses Mahalanobis metric, a simple, linear algebraic metric that was employed to generate fitness scores by

taking advantage of the sparsity of embeddings.

However, despite its usefulness, it still gave saturated RMSEs which implies that it is not perfect. But, it can serve as base for more complex and expensive fitness-scoring system which are employed at AI companies like OpenAI, Meta, Perplexity, etc.

6. Bibiliography

Bellet, A., Habrard, A., & Sebban, M. (2015). *Metric Learning*. Springer.