Machine Learning Developer Test Report

Introduction

This report describes the performance of two distance metrics (Cosine Distance and Euclidean Distance) for the classification of genetic syndromes using embeddings extracted from a KNN classification model. The goal is to evaluate the effectiveness of these methods using the AUC (Area Under the Curve) metric in a 10-fold cross-validation.

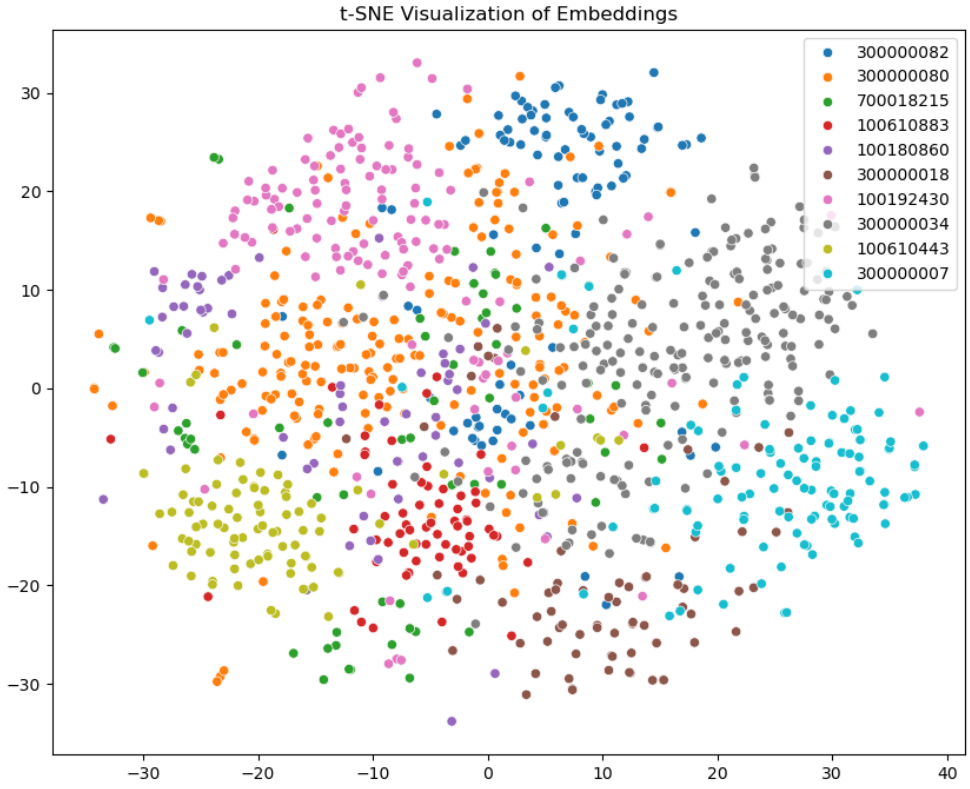
Methodology

The data consists of embeddings from a classification model for different genetic syndromes, stored in a pickle file. The dictionary structure is:

{'syndrome_id': { 'subject_id': {'image_id': [320x1 encoding]}}}

t-SNE Visualization

The purpose of applying t-SNE (t-Distributed Stochastic Neighbor Embedding) was to reduce the dimensionality of the embeddings and visualize the high-dimensional data in a 2D space. This helps in understanding how well-separated different genetic syndromes are and whether the embeddings from the model cluster in a meaningful way. The t-SNE plot of the embeddings is shown below.



Different colors represent different genetic syndromes. The presence of distinct clusters indicates that the embeddings capture differences between syndromes. Overlapping clusters suggest that the embeddings for these classes are similar in high-dimensional space. This might indicate that the classes share common features or are not distinctly separated by the model. However, there are some relatively well-separated classes, suggesting that the embeddings effectively differentiate between these classes.

Cross-Validation

The next step consists in compute Cosine and Euclidean distances between test vectors and gallery vectors from each test set of a 10 fold cross-validation, and then perform a KNN classification to evaluate the model based on ROC AUC scores in both methods.

Normalization is crucial in machine learning to ensure that features contribute equally to the model's performance. StandardScaler was chosen because it standardizes features by removing the mean and scaling to unit variance. This is particularly important when dealing with distance-based algorithms like KNN, where features need to be on a comparable scale to avoid biasing the distance computation towards features with larger scales. The parameters Mean = 0 and Standard Deviation = 1 ensures that the embeddings are centered around zero and have a standard deviation of one, which helps in achieving better model performance and convergence.

LabelEncoder was used to convert categorical syndrome IDs into numeric format. Most machine learning algorithms, including KNN, require numerical input, and encoding categorical labels is a common preprocessing step.

The number of neighbors is a crucial parameter for KNN. A smaller number of neighbors can lead to a model that is sensitive to noise (overfitting), while a larger number can smooth out the decision boundary (underfitting). n_neighbors=5 is a typical choice that provides a good trade-off between bias and variance.

The ROC AUC (Receiver Operating Characteristic - Area Under the Curve) is a performance metric used to evaluate the effectiveness of a classification model, especially in binary classification tasks. Here’s a breakdown of its key components and how to interpret it:

ROC Curve

X-Axis (False Positive Rate - FPR): This measures how often the model incorrectly classifies a negative class as positive (false positives). It’s calculated as:

FPR = False Positives / (False Positives + True Negatives)

Y-Axis (True Positive Rate - TPR or Recall/Sensitivity): This represents the proportion of actual positives that the model correctly identifies. It’s calculated as:

TPR = True Positives / (True Positives + False Negatives)

Curve: The ROC curve plots TPR against FPR at various threshold levels. It shows the trade-off between sensitivity and specificity across different decision thresholds.

AUC (Area Under the Curve)

The AUC represents the area under the ROC curve. It provides a single scalar value that summarizes the model's performance across all threshold levels.

Interpreting AUC Values

0.90 - 1.0: Excellent classification performance. The model is very good at distinguishing between classes.

0.80 - 0.90: Good performance. The model can classify well but may still make some mistakes.

0.70 - 0.80: Fair performance. The model is reasonably able to distinguish classes but could be improved.

0.60 - 0.70: Poor performance. The model is struggling to distinguish between the classes.

0.50 - 0.60: No real separation. The model performs close to random guessing.

Multi-Class and One-vs-Rest (OvR)

For multi-class classification, AUC is typically calculated in a one-vs-rest (OvR) approach, where each class is evaluated against all other classes. The final AUC is an average of these values, giving insight into how well the model distinguishes between each class versus all other classes.

Results

Cosine Distance

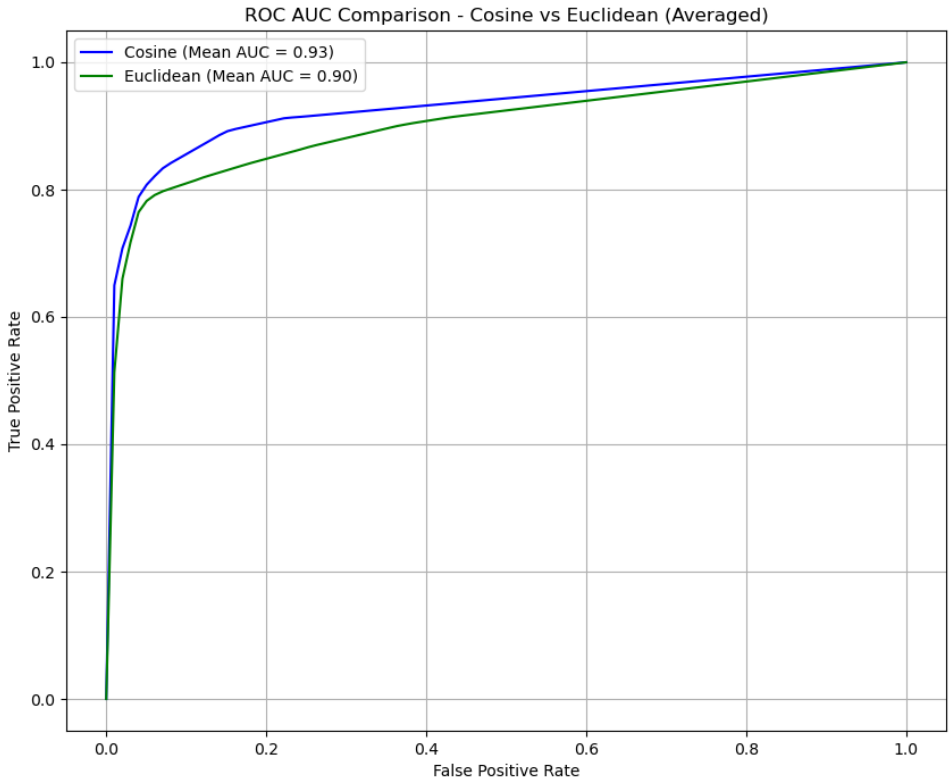
Fold	AUC
Fold 1	0.92
Fold 2	0.81
Fold 3	0.95
Fold 4	0.94
Fold 5	0.97
Fold 6	0.97
Fold 7	0.91
Fold 8	0.90
Fold 9	0.91
Fold 10	1.00

Average AUC: 0.93

Euclidean Distance

Fold	AUC
Fold 1	0.93
Fold 2	0.96
Fold 3	0.92
Fold 4	0.89
Fold 5	0.91
Fold 6	0.84
Fold 7	0.86
Fold 8	0.84
Fold 9	0.90
Fold 10	0.99

Average AUC: 0.91



The average AUC suggests that Cosine Distance has a slightly better performance compared to Euclidean Distance, with a higher average AUC score and more consistent results across folds.

- **Cosine Distance**: Demonstrates superior performance with higher average AUC and consistent results across folds.

- **Euclidean Distance**: Shows good performance but with more variability in results.

Based on the results, it is recommended to use cosine distance for the genetic syndrome classification task, as it provides more consistent and slightly better performance. Continue exploring model improvements and additional preprocessing techniques for potentially better results.

Reproduction Instructions

1. Set Up Environment:
    a. Ensure you have Python and necessary libraries installed (pickle, numpy, scikit-learn, matplotlib, seaborn, pytest).
2. Run tests:
    a. Use pytest to run the unit tests:
    `pytest test_functions.py`
3. Visualize the t-SNE plot::
    a. Run the script to generate the plot image:
    `python3 tsne_plot.py`
    b. The image `tsne_plot.png` will be generated.
4. Run the experiment:
    a. Run the experiment script:
    `python3 knn_evaluation.py`
    b. The image `roc_auc_plot.png` and the file `performance_comparison.txt` will be generated.