Machine Learning Junior Practical Test Report

1. Methodology

1.1 Data Processing

The dataset provided was a hierarchical pickle file containing embeddings derived from images. To prepare this data for analysis:

- **Loading** : I loaded the pickle file using Python's `pickle` module. This step was essential to ensure the data was ready for processing.
- **Transformation** : The hierarchical structure ('syndrome_id' → 'subject_id' → 'image_id') was converted into a tabular format, resulting in a DataFrame with the columns: `syndrome_id`, `subject_id`, `image_id`, and `embedding`. This approach facilitated data manipulation in subsequent steps.
- **Validation** :
  - I checked for missing or inconsistent data, ensuring the embeddings had the correct dimensionality (320 dimensions).
  - Duplicate entries were removed to ensure data integrity.
- **Output** : The processed data was saved as a CSV file for future use.

1.2 Exploratory Data Analysis (EDA)

- **Statistics** :
  - The dataset contained embeddings corresponding to **1,116 images** distributed across multiple `syndrome_id`s.
  - To better understand the sample distribution, I generated a bar chart showing the number of images per syndrome.
  - Significant imbalance was observed in some classes, with certain syndromes having fewer than 10 samples, potentially impacting classification results.

1.3 Data Visualization

- **t-SNE Visualization** :
  - The embeddings were reduced to two dimensions using t-SNE. This technique helped reveal patterns and clusters in the data.
  - I created:
    - A scatter plot with points colored by `syndrome_id`.
    - A density heatmap to highlight regions of high point concentration.
- **Observations** :
  - Some syndromes formed distinct clusters, indicating that the embeddings may have sufficient separability for classification.
  - However, I also identified overlapping clusters, suggesting that certain cases could pose challenges for the models.

1.4 Classification Task

- **K-Nearest Neighbors (KNN)** :
  - The use of KNN was specified in the test requirements, and it was implemented with Cosine and Euclidean distance metrics. However, in a practical scenario, I would also consider testing other models, such as neural networks or SVM, to evaluate if they could offer better results.
  - To determine the optimal value of `k` for the KNN model, I used cross-validation metrics via Scikit-learn's `cross_val_score` function. This approach allowed me to evaluate the model's performance across different `k` values (ranging from 1 to 15), using 10-fold cross-validation.
  - The criterion for selecting the best `k` was the **average accuracy obtained** across the folds. Each `k` value was tested with two distinct distance metrics (Euclidean and Cosine), and the `k` with the highest average performance was chosen as the optimal value for each distance metric.
  - This analysis helped ensure that the model was configured to maximize its generalization capability and avoid both underfitting and overfitting. The selected `k` values were documented in the result tables and significantly impacted the model's final performance.
- **Metrics** :
  - For the best models, I calculated the following metrics:
    - Accuracy
    - F1-Score
    - Top-k Accuracy
    - AUC (Area Under the ROC Curve)

    - Confusion matrices and classification reports were generated for both metrics.

1.5 Metrics Evaluation

- **ROC AUC Curves** :
    - I plotted ROC AUC curves for both metrics, calculating the average results across the folds of cross-validation.
    - Regarding the ROC curves, I observed very high values for both the Euclidean and Cosine AUCs (0.92 and 0.94, respectively). In a practical environment, I would investigate further to understand the reasons behind these high values, looking for any external factors or specific dataset characteristics that could explain them. However, since the other metrics also showed consistent and well-evaluated results, I considered that a detailed investigation was unnecessary for this test.
    - The curves were displayed together in a single graph for easy comparison.
- **Summary Tables** :
    - I generated a CSV file summarizing the performance metrics for both distance metrics. This table included Accuracy, F1-Score, Top-k Accuracy, and AUC.
    - Performance_metrics

2. Results

2.1 Data Processing Results

- Total samples after processing: **1,116** .
- Number of unique syndromes: **10** .
- Distribution of images per syndrome saved as `syndrome_distribution.png`.

2.2 Visualization Results

- t-SNE Scatter Plot: Saved as `tsne_visualization.png`.
- t-SNE Density Heatmap: Saved as `tsne_cluster_density_heatmap.png`.

2.3 Classification Results

Performance Metrics Summary

Metric	Euclidean Distance	Cosine Distance
Accuracy	0.829	0.863
Top-k	15	12
F1-Score	0.827	0.862
AUC	0.985	0.990

Confusion Matrices

- Saved as:
  - `confusion_matrix_euclidean.png`
  - `confusion_matrix_cosine.png`
- The confusion matrices showed satisfactory results, indicating that the model does not face significant difficulties in correctly identifying the classes and minimizing false positives or false negatives. For a more detailed and robust evaluation, it would be ideal to include new data inputs to further validate the model's performance in varied scenarios.

Classification Reports

Euclidean Distance:

```
              precision    recall  f1-score   support

   100180860       0.75      0.85      0.80        67
   100192430       0.70      0.91      0.79       136
   100610443       0.91      0.91      0.91        89
   100610883       0.72      0.89      0.79        65
   300000007       0.87      0.90      0.88       115
   300000018       0.81      0.65      0.72        74
   300000034       0.85      0.91      0.88       210
   300000080       0.89      0.77      0.83       198
   300000082       0.92      0.81      0.86        98
   700018215       0.94      0.52      0.67        64

    accuracy                           0.83      1116
   macro avg       0.84      0.81      0.81      1116
weighted avg       0.84      0.83      0.83      1116
```

Cosine Distance:

	precision	recall	f1-score	support
100180860	0.78	0.88	0.83	67
100192430	0.85	0.85	0.85	136
100610443	0.99	0.98	0.98	89
100610883	0.86	0.75	0.80	65
300000007	0.85	0.92	0.88	115
300000018	0.83	0.77	0.80	74
300000034	0.89	0.95	0.92	210
300000080	0.85	0.85	0.85	198
300000082	0.85	0.84	0.84	98
700018215	0.87	0.64	0.74	64
accuracy			0.86	1116
macro avg	0.86	0.84	0.85	1116
weighted avg	0.86	0.86	0.86	1116

3. Analysis

3.1 Observations from t-SNE

- Distinct clusters indicate potential separability for some syndromes.
- Overlapping clusters suggest possible challenges in classifying syndromes with similar embeddings.

3.2 Comparison of Distance Metrics

- **Cosine Distance** slightly outperformed **Euclidean Distance** across all metrics (Accuracy, F1-Score, AUC, Top-k Accuracy).
- The performance difference may be attributed to:
  - Cosine distance’s ability to handle high-dimensional data where the magnitude of vectors may matter less than their direction.

3.3 Challenges and Solutions

- **Challenge** : Class imbalance.

- **Solution** : Increase training data for underrepresented classes or apply class weights in future iterations.
- **Challenge** : Overlapping clusters in t-SNE visualization.
- **Solution** : I would have used more advanced models (e.g., SVM, neural networks) in complementary experiments to handle complex decision boundaries.
- **Challenge** : Hyperparameter tuning in KNN and other models.
- **Solution** : I would implement a systematic hyperparameter search (e.g., Grid Search or Random Search) to find the best values for k and parameters related to distance metrics, maximizing the model's performance.
- File handling: Ensured an appropriate directory structure to avoid saving errors.
- Metric inconsistencies: Verified metric calculations and cross-validation setups.

4. Recommendations

- Experiment with other classifiers (e.g., Random Forest, SVM).
- Address class imbalance with data augmentation or oversampling techniques.
- Perform hyperparameter optimization for KNN and other models.
- Investigate the embeddings more deeply (e.g., PCA, clustering).

5. Conclusion

- I observed that Cosine distance was slightly more effective for this task, particularly in metrics like AUC and F1-Score. This suggests that models prioritizing the direction of embeddings rather than their magnitude might be better suited for high-dimensional data like this.
- During the process, I identified challenges related to class imbalance and overlapping clusters in the data. These factors can directly influence the model's ability to generalize to new data and require greater attention in future projects.
- Despite these challenges, the overall results indicate that the implemented pipeline serves as a promising foundation for genetic syndrome classification, providing valuable insights into the data and model performance.
- In a potential real-world application, I would explore the data distribution more deeply and apply techniques to mitigate the impact of underrepresented classes, such as oversampling or adaptive class weighting.
- I also consider it important to evaluate more complex models, such as deep neural networks, which could capture more sophisticated patterns in the

embeddings and potentially improve accuracy and generalization.

6. References

- Scikit-learn documentation.
- Python Matplotlib and Seaborn guides for visualization.