ML Junior Practical Test - Report

Genetic Syndrome Classification with KNN

1. Methodology

1.1 Data Preprocessing

- **Loading the file (.p)**: I started by reading the `mini_gm_public_v0.1.p` file, which contains a hierarchical dictionary with 320-dimension embeddings.
- **Hierarchical Structure → Flatten**: I transformed the nested structure (syndromeId → subjectId → imageId → embedding) into a DataFrame, where each row represents an image, containing the columns `syndromeId`, `subjectId`, `imageId` and `embeddingVector`. • **Checking missing data**: I did a simple check to confirm there were no null values. As the console image shows, there was no missing data in `syndromeId`, `subjectId`, `imageId` or `embeddingVector`.

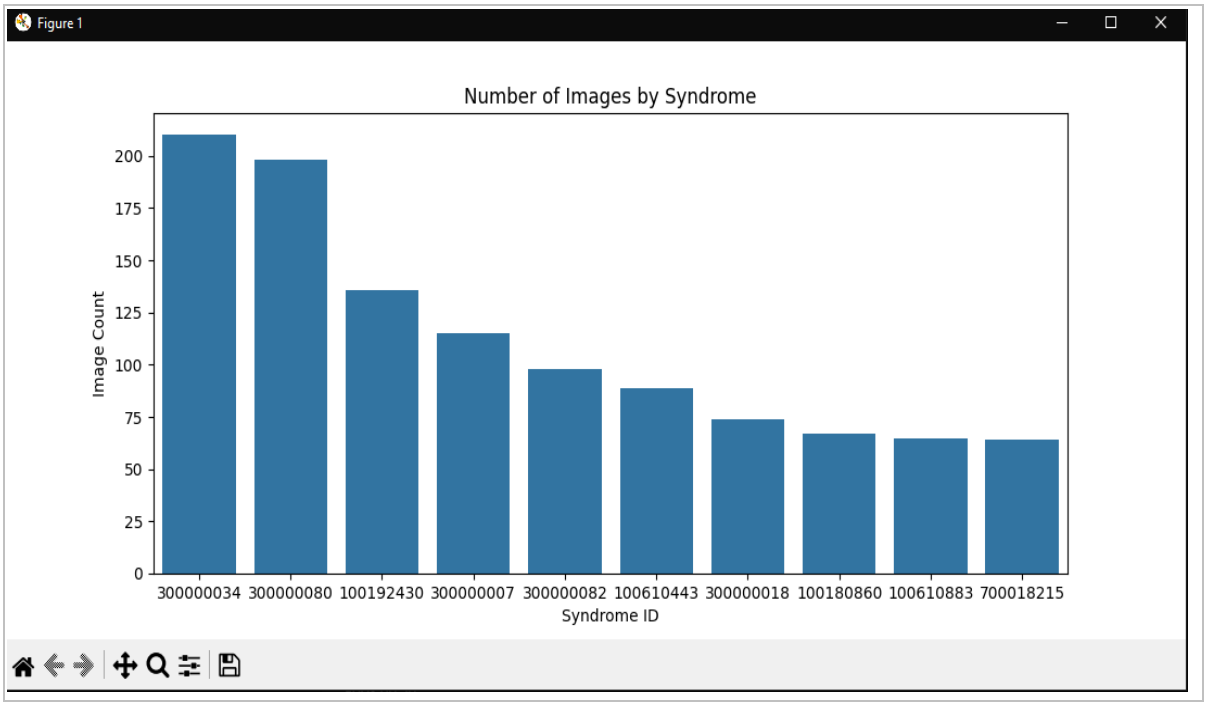
Figure: Distribution of missing data and general statistics in console:

```
Missing data in each column:
syndromeId        0
subjectId         0
imageId           0
embeddingVector   0 dtype:
int64

Total number of samples: 1116
Number of unique syndromes: 10
Samples per syndrome (min to max): 64 - 210 X
shape: (1116, 320) | y shape: (1116,)
```

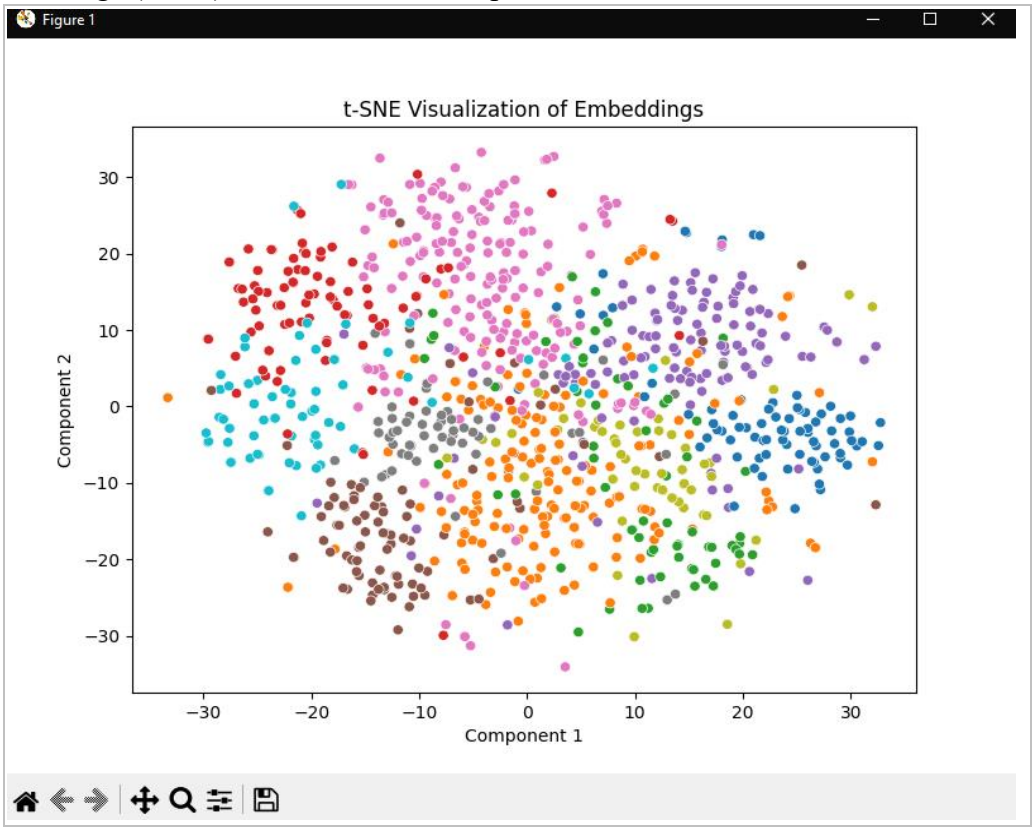
1.2 Exploratory Data Analysis (EDA)

- **Class Distribution**: I plotted a bar chart to see how many images exist for each syndrome. The following figure shows that `syndromeId=300000034` has the highest number of images (~210), followed by `300000080`, while others have fewer, like `700018215`.



1.3 Visualization with t-SNE

- To better understand the separation between classes, I applied t-SNE reducing the embeddings (320D) to 2D. Each color represents a `syndromeId`.



We noticed some more cohesive groups, but also areas where colors mix, which indicates that some syndromes might be more similar in the embedding space.

1.4 Classification with KNN

- **Algorithm**: I chose **K-Nearest Neighbors**, comparing two distance metrics: Euclidean and Cosine.
- **Range of k**: I evaluated k values from **1 to 15**.
- **10-Fold Cross-Validation**: I used stratified validation, ensuring each fold respected the class proportions.

Results of F1-Score for each k

Below are the resulting tables (printed in console with `tabulate`):

```
F1-scores (Euclidean)
+----+-----------+
| k  | F1-Score  |
|----+-----------|
| 1  | 0.6307    |
| 2  | 0.5875    |
| 3  | 0.6333    |
| 4  | 0.6580    |
| 5  | 0.6659    |
| 6  | 0.6734    |
| 7  | 0.6879    |
| 8  | 0.7030    |
| 9  | 0.7070    |
| 10 | 0.6980    |
| 11 | 0.7102    |
| 12 | 0.7070    |
| 13 | 0.7234    |
| 14 | 0.7323    |
| 15 | 0.7346    |
+----+-----------+
Best k (Euclid): 15

F1-scores (Cosine)
+----+-----------+
| k  | F1-Score  |
|----+-----------|
| 1  | 0.6755    |
| 2  | 0.6561    |
| 3  | 0.7148    |
| 4  | 0.7298    |
| 5  | 0.7559    |
| 6  | 0.7658    |
| 7  | 0.7794    |
| 8  | 0.7741    |
| 9  | 0.7703    |
| 10 | 0.7580    |
| 11 | 0.7568    |
| 12 | 0.7599    |
| 13 | 0.7620    |
| 14 | 0.7563    |
| 15 | 0.7660    |
+----+-----------+
Best k (Cosine): 7
```

Therefore, the **best k** for Euclidean is **15**, while for Cosine it's **7**.

2. Results

2.1 Detailed Metrics

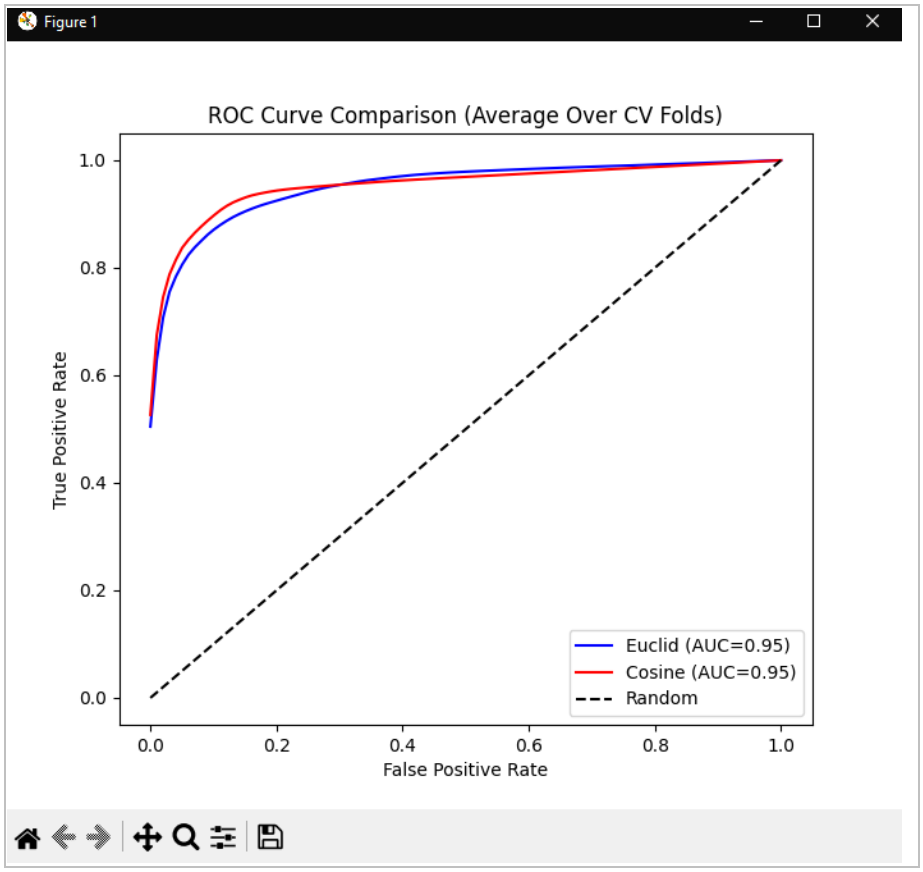
For each distance measure, I also calculated statistics such as average F1 (`meanF1`), standard deviation of F1 (`stdF1`), Top-5 Accuracy (`meanTop5`) and AUC (`meanAUC`).

```
=== Detailed Stats (Euclidean) ===
+-----------+-------+
| Metric    | Value |
|-----------+-------|
| meanF1    | 0.7346|
| stdF1     | 0.0395|
| meanTop5  | 0.9677|
| meanAUC   | 0.9504|
| stdAUC    | 0.0103|
+-----------+-------+

=== Detailed Stats (Cosine) ===
+-----------+-------+
| Metric    | Value |
|-----------+-------|
| meanF1    | 0.7794|
| stdF1     | 0.0469|
| meanTop5  | 0.9668|
| meanAUC   | 0.9528|
| stdAUC    | 0.0150|
+-----------+-------+
```

2.2 ROC Curves Comparison

To generate the ROC curves, I aggregated the probabilities from each fold (One-vs-Rest) and calculated the average TPR/FPR. I then plotted the Euclid (blue) and Cosine (red) curves on the same graph:



We can see that the red curve (Cosine) generally stays above the blue one (Euclid), although the final AUC is very similar (both ~0.95). This indicates that Cosine performed slightly better, but the difference wasn't huge in the average of the folds.

3. Analysis

1. **Difference Between Metrics**:
    - **Cosine** worked a bit better, possibly because the 320-dimension embeddings benefit more from the angle than from the magnitude of the vectors. Euclidean
    - needed a higher k (15) to stabilize.
2. **Top-5 Accuracy**:
    - Both showed values above 0.96, indicating that the real syndrome usually appeared among the 5 most likely predictions from the model.
3. **Class Distribution**:
    - Some syndromes had many more images than others (e.g., 210 vs. 64). This can impact the model's stability, but cross-validation helped mitigate this imbalance.

4. **t-SNE Visualization**:
    - The graph showed moderate clusters, but various colors mixed together, explaining why the model doesn't reach 100% F1.

4. Challenges and Solutions

1. **Hierarchical Structure**
    - *Challenge*: The dataset didn't come in a standard CSV format, but as a nested dictionary.
    - *Solution*: I implemented a flatten function, generating a DataFrame (one "record" per image).
2. **Execution Time**
    - *Challenge*: Running cross-validation (k=1..15 and 10 folds) and t-SNE (320D → 2D) can be time-consuming.
    - *Solution*: I limited t-SNE to sample only 1000 points and kept 10 folds as it's a reliable standard for validation.
3. **Multiclass Evaluation**
    - *Challenge*: Plotting ROC curves in multiclass mode requires binarizing each class (One-vs-Rest).
    - *Solution*: I used functions with `label_binarize` and aggregated the TPR/FPR from each fold, creating an "average curve."

5. Recommendations

1. **Embedding Normalization**: Trying to normalize each vector to have norm 1 could further improve the Cosine distance performance.
2. **Test Other Classifiers**: Random Forest, SVM, or even a simple neural network might outperform KNN on larger datasets.
3. **Class Balance**: If possible, seek more images of syndromes with few samples, or use oversampling techniques.
4. **Deeper Hyperparameter Tuning**: Adjust weights for each class in KNN or expand the search for k, and use weights (e.g., `weights='distance'`).
5. **Production Use**: If there are plans to put this model into operation, we could create a microservice (Flask/FastAPI) and monitor metrics on real data.

Conclusion

This project demonstrated a **complete pipeline** for syndrome classification from embeddings, covering:

- Loading and flattening of the hierarchical dataset;
- Exploratory data analysis (EDA) and visualization with t-SNE;
- Comparison of KNN using Euclidean vs. Cosine distances, varying k from 1 to
- 15; Evaluation with F1-Score, multiclass AUC, and Top-5 Accuracy; Generation
- of tables and graphs to better interpret the results.

Overall, KNN with **Cosine** distance and **k=7** achieved the best F1 score (approximately 0.78), while the Euclidean option needed k=15 to reach ~0.73. Nevertheless, both had AUC close to 0.95, indicating that the embeddings are expressive. In future work, I can try normalizing vectors, adding more data from underrepresented syndromes, and testing more advanced classifiers to further improve accuracy.