Classification of Syndrome IDs Using Image Embeddings

Methodology

Data Processing

We started with a pickle file (`mini_gm_public_v0.1.p`) containing 320-dimensional image embeddings in a nested structure: syndromes → subjects → images. Here's how we prepped it:

- **Loading**: Opened the file with Python's `pickle` module in binary mode to avoid encoding errors.

- **Flattening**: Turned the nested dictionary into simple lists—embeddings (images), syndrome IDs, subject IDs, and image IDs—for easier analysis.

- **Integrity**: Checked for missing values (none found) and ensured all embeddings were 320D (all matched). Saved this clean data as `flattened_data.pkl`.

Exploratory Data Analysis (EDA)

We dug into the dataset to understand it better: - Counted total images, syndromes, and subjects. - Broke down images and subjects per syndrome, plus stats on images per subject. - Looked at embedding properties (mean, variance) and imbalances.

Data Visualization

To see the embeddings visually: - Used t-SNE to shrink 320D embeddings to 2D. - Plotted them with colors for each syndrome, saved as `tsne_plot.png`. - Noted patterns and their impact on classification.

Classification Task

For classifying syndromes: - Chose KNN (K-Nearest Neighbors) with Cosine and Euclidean distances—Cosine for direction, Euclidean for distance. - Ran 10-fold cross-validation to test (k) from 1 to 15, picking the best based on accuracy. - Measured AUC, F1-Score (macro), and Top-3 Accuracy per fold.

Metrics and Evaluation

To wrap up: - Generated ROC curves for both metrics, averaged across folds, and plotted them in `roc_curves.png`. - Summarized metrics (AUC, F1, Top-3) in a text table, saved as `knn_metrics_summary.txt`.

All steps were coded in `X.py` with dependencies in `requirements.txt` (numpy, scikit-learn, matplotlib).

Results

Exploratory Data Analysis

From `eda_summary.txt`: - **Total Images**: 1116 - **Unique Syndromes**: 10 - **Unique Subjects**: 941 - **Images per Syndrome**: Ranged from 64 (700018215) to 210 (300000034). - **Images per Subject**: Average 1.19, median 1, max 11. - **Subjects per Syndrome**: 59 (100610883) to 180 (300000080). - **Embedding Variance**: All 320 dimensions had variance > 0.01. - **Imbalance Ratio**: 3.28 (210 / 64), showing significant imbalance.

Data Visualization

From `tsne_observations.txt` and `tsne_plot.png`: - Points were "pretty tight" suggesting overlap between syndromes in 2D. - Each syndrome had a unique color, but separation wasn't clear.

Classification Task

From `knn_results.txt`: - **Cosine Distance (k=11)**: - Best Accuracy: 0.790 - Average AUC: 0.960 - Average F1-Score: 0.751 - Average Top-3 Accuracy: 0.933 - **Euclidean Distance (k=15)**: - Best Accuracy: 0.743 - Average AUC: 0.948 - Average F1-Score: 0.706 - Average Top-3 Accuracy: 0.913

Metrics and Evaluation

- **ROC Curves**: See `roc_curves.png`—Cosine (AUC 0.960) slightly above Euclidean (AUC 0.948).
- **Metrics Table**: Saved in `knn_metrics_summary.txt`, matching the above results.

Analysis

Performance Comparison

- **Cosine Outperforms Euclidean**: Higher accuracy (0.790 vs. 0.743), AUC (0.960 vs. 0.948), F1 (0.751 vs. 0.706), and Top-3 (0.933 vs. 0.913).

- **Why?**: Cosine focuses on direction, which seems to capture syndrome differences better in these embeddings. Euclidean, caring about magnitude, might be less effective if magnitude varies within syndromes but not between them.

- **t-SNE Insight**: Tight 2D clusters suggested overlap, yet Cosine's high AUC implies the 320D space still separates syndromes well t-SNE might've missed higher-dimensional distinctions.

Insights

- **Strong Separation**: AUCs near 0.95-0.96 show the embeddings are solid for classification, despite tight t-SNE points.

- **Imbalance Impact**: F1-Scores (0.751, 0.706) lag behind accuracy, likely due to the 3.28 imbalance ratio—minority syndromes (e.g., 700018215 with 64 images) might be harder to predict.

- **Top-3 Success**: 93%+ Top-3 accuracy means even if KNN misses the exact syndrome, it's usually close, useful for practical applications.

Recommendations

1. **Tune Embeddings**: If tight clusters persist, try PCA or feature selection on the 320D embeddings to boost separation.

1. **Handle Imbalance**: Add `class_weight="balanced"` to KNN or use SMOTE to even out syndrome counts.
2. **Try Other Models**: Test SVM or neural networks—might catch patterns KNN misses.
3. **More Data**: With only 1.19 images per subject, more images could improve robustness.
4. **Expand Metrics**: Add precision/recall per syndrome to pinpoint weak classes.

This project's a strong start—Cosine KNN at 79% accuracy is promising for 10 classes!