Genetic Syndromes Report

1. Introduction

This report presents the implementation and evaluation of a machine learning pipeline for classifying genetic syndromes using image embeddings. This report is divided between this introduction, the methodology, main results, challenges and solutions, and, finally, recommendations.

2. Methodology

a. Data Preprocessing

The syndrome classification pipeline begins with comprehensive data processing to prepare the raw data for modeling:

The raw data was consisted by a hierarchical pickle file with the image ID, syndrome ID, subject ID, and the image array embedded. To explore the data, it was necessary to manipulate it to a Pandas DataFrame. During this transformation, the data was cleaned, firstly flattened, after removing some inconsistences, as missing values (which weren't found), or embeddings with different lengths, the normalization was applied to ensure that all image features contribute equally in order to satisfy the main algorithm's metric: the distance calculation.

The data splitting was performed, dividing 20% of data to be the test set, and 80% to be the training one, being the X (features) to make the model' predictions the embeddings itself, and the syndrome ID the label (Y).

b. Exploratory data analysis

This step was made also using Pandas library and Matplotlib. Here the images quantity per syndrome was calculated and analyzed, as well as the number of individuals per syndrome. In addition, the number of instances per syndrome, to check if there is any unbalance between the classes.

c. Model classification

As ordered, the KNN was performed using a cross validation with 10 K-fold to find the best number of neighbors, using both distance metrics (Euclidean and Cosine), and the accuracy as main metric to compare the models. It was also calculated the F1 score and the Top K Accuracy, being K equals 3. After this, the best model of each distance metric was selected to be retrained and tested.

d. Evaluation metrics

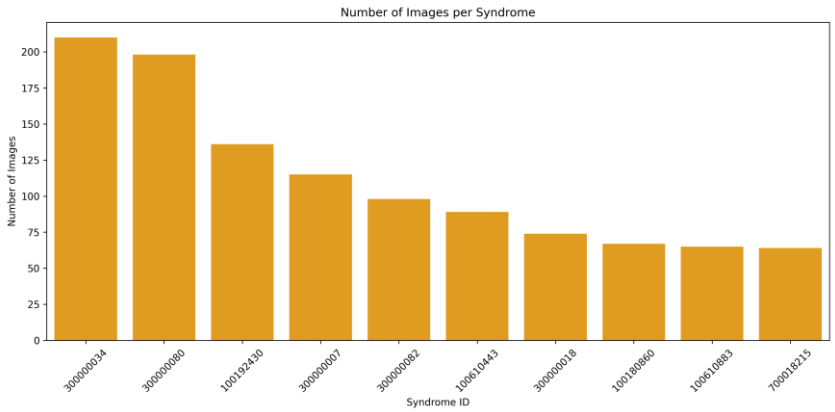
Having the best models for each distance metric, all the requested metrics was calculated: F1-score, accuracy and AUC ROC Curve. All of them for both training and testing sets, and the ROC for each of the syndromes.

3. Results

a. Data

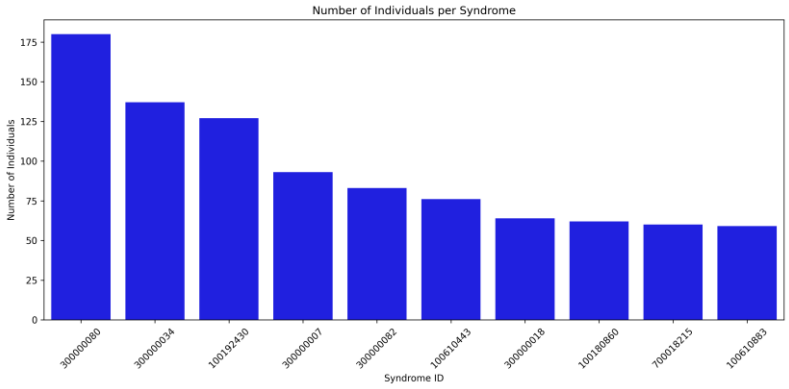
Here follows some analysis regarding to the complete preprocessed data.

Figure 1: Number of Images per Syndrome



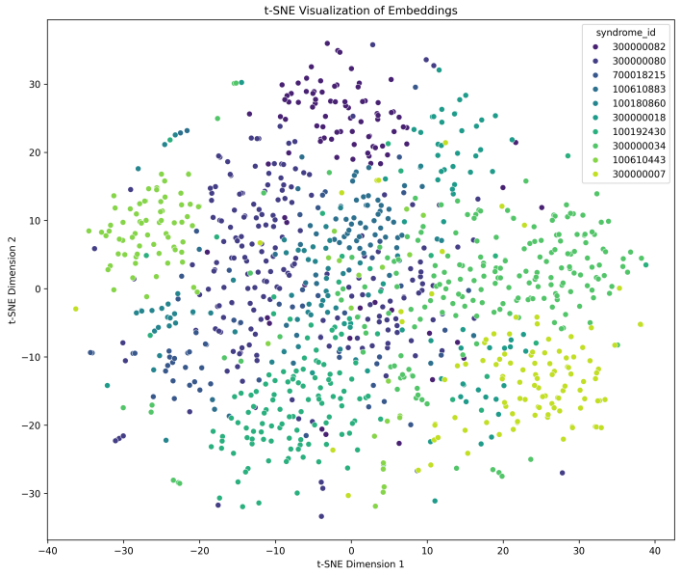
Looking the Figure 1, it can be observed 10 types of syndromes in this dataset, and their samples are not equals. Having more instances for some of the syndromes, so that ones with less samples might have worst classification performance.

Figure 2: Number of Individuals per Syndrome



The Figure 2 also shows a similar behavior, which shows that different syndromes does have more individuals with it than others. However, the biggest quantities are not for the same syndrome ID that it brings Figure 1. This might also impact in the model's learning, because it can only memorize specific phases, and also those frequent syndromes might overfit the model.

Figure 3: T-SNE of embeddings



Analyzing Figure 3, we can see each embedding, having the color that belongs to a class (syndrome), we can some samples more distant from the center, where it has a bigger concentration of instances, as 300000082 and 100610683, which are very close from each other.

In this graphic, it can also be seen that some points are more distant, indicating maybe an outlier, or with more samples of it, a new group.

As mentioned before, the data was divided between train and test, having 892 instances and 224, respectively.

b. Model classification

Figure 4: Table with 20 best models

	distance	n_neighbors	accuracy_mean
0	cosine	12	0.785843
1	cosine	11	0.785818
2	cosine	9	0.783658
3	cosine	10	0.782497
4	cosine	13	0.782472
5	cosine	15	0.781411
6	cosine	14	0.779089
7	cosine	8	0.774707
8	cosine	7	0.774694
9	cosine	5	0.767915
10	cosine	6	0.758926
11	cosine	4	0.742135
12	cosine	3	0.735543
13	euclidean	15	0.734332
14	euclidean	14	0.730874
15	euclidean	13	0.728677
16	euclidean	12	0.720849
17	euclidean	10	0.718664
18	euclidean	8	0.718602
19	euclidean	9	0.718589

Figure 4 shows the best KNN models obtained in the cross validation (CV). It observed that using Cosine had better mean accuracy than Euclidian distance for almost all K values. The better Cosine distance model was using 12 neighbors, achieving 0.79 of mean accuracy in the CV, while the best Euclidean distance was using 15 neighbors, resulting in 0.73 as mean accuracy, and one of the "worsts" of Cosine was with 3 neighbors, having approximately 0.74 in this metric.

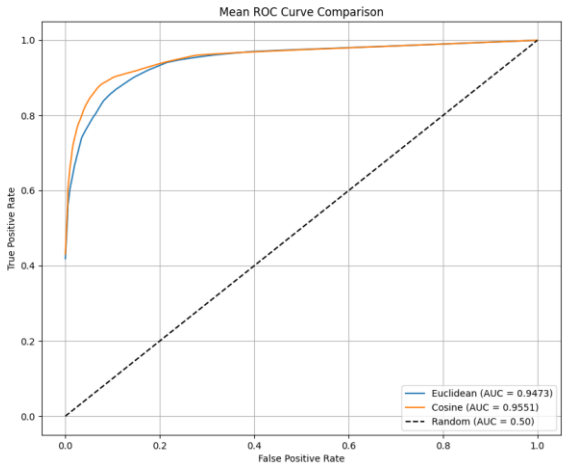
Figure 5: Table with best model of each metric

	distance_metric	dataset	accuracy	f1_score	top3_accuracy	n_neighbors
0	cosine	train	0.837444	0.816742	0.965247	12
1	euclidean	train	0.782511	0.747891	0.951794	15
2	cosine	test	0.781250	0.743895	0.924107	12
3	euclidean	test	0.718750	0.689416	0.906250	15

In Figure 5, it has the best model's results for each metric after training apart. The Cosine's model achieved again better performance values in training and testing, getting 0.84, 0.82 and 0.97 in, respectively, Accuracy, F1 Score and Top-K accuracy during the training, and 0.78, 0.74 and 0.92 in testing. The Euclidean model got 0.78, 0.75 and 0.95 for its training, and 0.72, 0.69, 0.91 in testing. So, the best metric in each case was the Top-K accuracy, especially for the training.

Despite of the good performance, it can be observed an overfitting between both models, already the training achieved far and bigger metric values than in the testing. Proving maybe a necessity to re-split the data to check if change the classes balances, the learning would be improved.

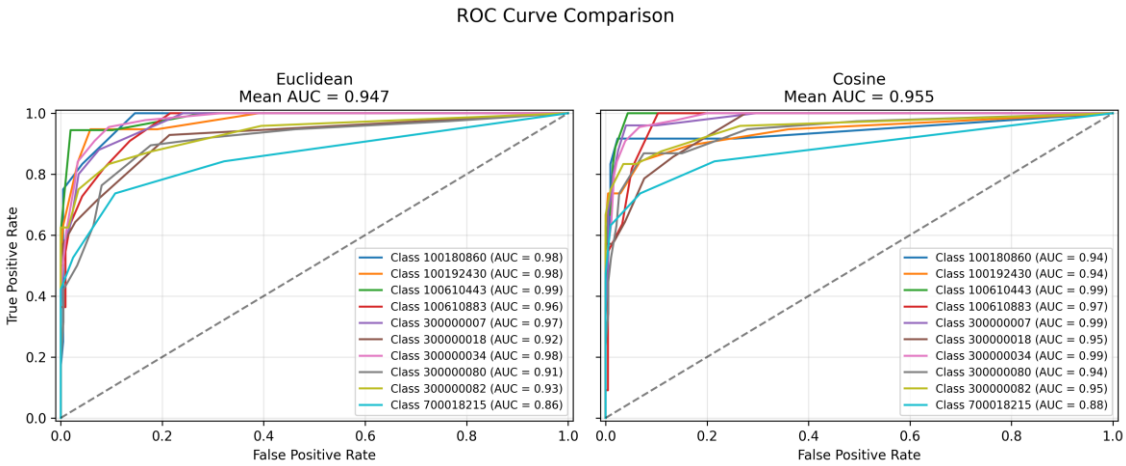
Figure 6: Mean AUC ROC curves



Regarding to the AUC ROC curve, in Figure 6, it has the mean values for each distance metric best model, where both have similar curve after 0.2 as False Positive Rate (FPR), and before this value, the Euclidean model shows lower True Positive Rate (TPR)

than the Cosine one, having an AUC of approximately 0.95 for Euclidean, and 0.95 for Cosine. In general, this graph shows that those models performed well, already high AUCs indicates better discrimination capacity of the model, and in both distance, this is indicated. However, again Cosine model performed better than Euclidean ones.

Figure 7: AUC ROC curves per class



Regarding Figure 7, it shows the AUC ROC Curve for each syndrome in both models. It can be realized that in all distances, the class 700018215 had the worst result, having the curve below all of the other classes, and an AUC of 0.88 in Cosine and 0.86 in Euclidean. This was one of the classes with less samples in the data. Already the syndromes in more individuals, like 300000080, also didn't performed so well. Some classes, like 300000034 and 100610443 had high precision, with the highest AUC: 0.98 and 0.99 in Euclidean, and 0.99 and 0.99 in Cosine. These syndromes didn't have the biggest quantity of instances in the data, suggesting that the data splitting was good for those classes, and bad for others. So, it can be seen again, the necessity to rebalance and re-split the data in order to make sure that all classes have similar instance size in all sets

So, looking all graphics and results found, the best model achieved was the KNN using Cosine as distance metric, with 12 neighbors, and 224 and 892 instances for training and testing.

4. Challenges and Solutions

The first one was to understand the data, and its relationship, which was solved as the preprocessing as done, as the exploration and visualization analysis. The implementation of the cross validation looking for saving all scores for each K used, the Gird Search method wasn't implemented because, only the final mean would be gotten, and it was necessary also having all K values. So, only the cross-validation method was used in this part.

To implement all AUC ROC curves plotting and calculation was also a challenge to face, joining to structure the code. To do this, it was used some tools to clean the code, like the GitHub copilot and Sourcery extensions.

The time to organize, implement, test and analyze was also difficult, especially to test other insights, like a performance comparison between balanced and not forced balanced data models.

5. Recommendations

For the model, it would be a good strategy to check a stratified training and test split, and also check the data balance after those parts. Try other techniques to handle the class imbalances, like data augmentation, and maybe another optimization for feature selection.