Apollo Solutions Machine Learning Developer Test

Lilian Iazzai de Souza Oliveira

1. Methodology

1.1. Data Preprocessing and Exploratory Analysis

```
file_path = "mini_gm_public_v0.1.p"

with open(file_path, "rb") as f:
    data = pickle.load(f)
rows = []

for syndrome_id, subjects in data.items():
    for subject_id, images in subjects.items():
        for image_id, embedding in images.items():
            rows.append({
                "syndrome_id": syndrome_id,
                "subject_id": subject_id,
                "image_id": image_id,
                "embedding": embedding
            })

df = pd.DataFrame(rows)
```

The first thing done was uploading the pickle file and transforming it into a tabular format, to enhance data accessibility, flexibility, and compatibility.

```
print(df.head())
print("Dataset Informations:\n", df.info()) # It doesn't have NaN values
# Changing datatypes
df['syndrome_id'] = df['syndrome_id'].astype('category')
df['image_id'] = pd.to_numeric(df['image_id'])
df['subject_id'] = pd.to_numeric(df['subject_id'])

print("Size of embeddings:\n", df['embedding'].apply(len).unique())
df = df.drop_duplicates(subset=['syndrome_id', 'subject_id', 'image_id'])

# Veifying imbalances
df['syndrome_id'].value_counts().plot(kind='bar')
plt.title('Distribution of Syndromes')
plt.xlabel('syndrome_id')
plt.ylabel('Contagem')
plt.show()

print("Images per Syndrome\n", df['syndrome_id'].value_counts())

# Converting: embeddings -> numpy arrays
X = np.array(df["embedding"].tolist())
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df["syndrome_id"])
```

- The method head() was used to visualize the dataset and see how it turned out.
- All the datatypes were “object”, then they were converted to their respective data types (categorical, numerical and numpy array) to improve data integrity. It will also help later when implementing KNN.
- The size of embeddings were verified to guarantee that all of them were 320-dimensional. Also, possible duplicate values were dropped.
- A chart showing the data distribution was plotted to see if there were any imbalances: there were types of syndrome with much more images than others, and it may affect the performance of a classification model.
- Then, the target variable (syndrome_id) was separated from the embeddings that will be used later as output and input of a classification model.

1.2. Dimensionality Reduction and Visualization (t-SNE)

```
palette = px.colors.qualitative.Safe
tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X)

fig = px.scatter(
    x=X_tsne[:, 0],
    y=X_tsne[:, 1],
    color=y,
    color_discrete_sequence=palette
)
fig.update_layout(
    title="2D Embeddings (t-SNE)",
    xaxis_title="First t-SNE",
    yaxis_title="Second t-SNE",
    title_font=dict(size=20, family="Arial", color="darkblue"),
    paper_bgcolor="white",
    plot_bgcolor="rgba(240, 240, 240, 0.8)",
    font=dict(size=14),
    margin=dict(l=20, r=20, t=40, b=20),
)

fig.update_traces(marker=dict(size=8, line=dict(width=1, color="black")))
fig.show()
```

The method t-SNE was used to reduce the dimensionality to 2D, making complex high-dimensional data more interpretable and actionable. Once this was done, a chart was created to visualize data, find patterns and detect outliers.

1.3. K-Nearest Neighbors

```
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X, y)

scalers = {'StandardScaler': StandardScaler(), 'Normalizer': Normalizer()}
datasets = {'Balanced': (X_resampled, y_resampled), 'Unbalanced': (X, y)}
i_values = [i for i in range(1, 16)]
metrics = {'euclidean', 'cosine'}
results = {'accuracy': {}, 'roc_auc': {}, 'f1_score': {}, 'top_k': {}}

# Cross-Validation
for dataset_name, (X_data, y_data) in datasets.items():
    for scaler_name, scaler in scalers.items():
        for metric in metrics:
            acc_scores = []
            f1_scores = []
            top_k_scores = []

            for i in i_values:
                pipeline = Pipeline([
                    ('scaler', scaler),
                    ('knn', KNeighborsClassifier(metric=metric, weights='distance', n_neighbors=i))
                ])
```

SMOTE (Synthetic Minority Over-sampling Technique) was used to oversample the minority class, to balance the dataset by generating synthetic samples and improve the model's ability to learn from the underrepresented class. This approach reduces bias toward the majority class, leading to better generalization and more accurate predictions for the minority class without losing valuable information from the original data.
Then, a loop was used to iterate through the datasets (one balanced and one unbalanced), types of scalers (StandardScaler and Normalizer) and metrics (euclidean and cosine), to test the model (K-Nearest Neighbors) with different values.

The number of nearest neighbors were ranging from 1 to 15, to check which one of them would give the best result.
This algorithm was tested with and without the "weight" parameter, and with different values in it ('distance' or 'uniform'). The best results were given with weight = 'distance'.
10-fold cross-validation was used to ensure a more reliable and robust evaluation of the model by splitting the data into 10 subsets and training/testing the model on different combinations. This technique provides a better estimate of model performance, reduces overfitting, and ensures that the model generalizes well to unseen data.

```
                acc_temp.append(pipeline.score(X_test, y_test))
                f1_temp.append(f1_score(y_test, y_pred, average="macro"))
                top_k_temp.append(top_k_accuracy_score(y_test, y_pred_proba, k=5))

            acc_scores.append(np.mean(acc_temp))
            f1_scores.append(np.mean(f1_temp))
            top_k_scores.append(np.mean(top_k_temp))
```

Inside that loop, metrics of evaluation, such as accuracy, F1-Score and Top-k, were calculated to see the performance of each variation of KNN. Each model has its own chart to compare the results between these metrics.

1.4. ROC Curve

```
y_binarized = label_binarize(y, classes=np.unique(y))
n_classes = y_binarized.shape[1]

fpr = {i: [] for i in range(n_classes)}
tpr = {i: [] for i in range(n_classes)}
roc_auc = {i: [] for i in range(n_classes)}

# Iterating through folds
for train_index, test_index in kf.split(X_scaled, y):
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y_binarized[train_index], y_binarized[test_index]

    # Training
    knn.fit(X_train, np.argmax(y_train, axis=1))
    y_score = knn.predict_proba(X_test)

    # Calculating ROC for each one
    for i in range(n_classes):
        fpr_temp, tpr_temp, _ = roc_curve(y_test[:, i], y_score[:, i])
        fpr[i].append(fpr_temp)
        tpr[i].append(tpr_temp)
        roc_auc[i].append(auc(fpr_temp, tpr_temp))

mean_fpr = np.linspace(0, 1, 100)
plt.figure(figsize=(10, 8))
```

After all of this, the last step of this code was to generate the Receiver Operating Characteristic curve. Using the ROC AUC curve helps to assess the model's performance across all classification thresholds. Provides a clear measure of how well the model distinguishes between classes, helping to evaluate the model's effectiveness, especially on imbalanced datasets.
This was a multiclass problem, so it was necessary to binarize each class in it. This chart was built for the two models, one euclidean and another cosine.

2. Results

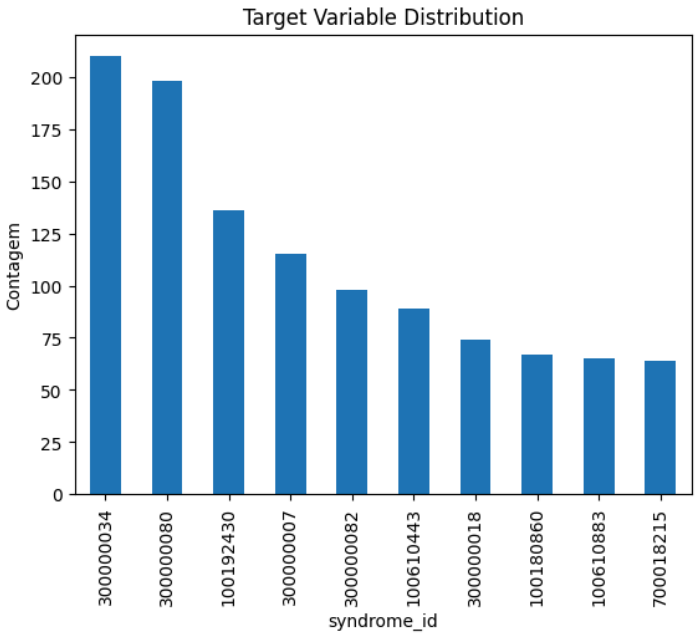

Chart 1 - Unbalanced Data

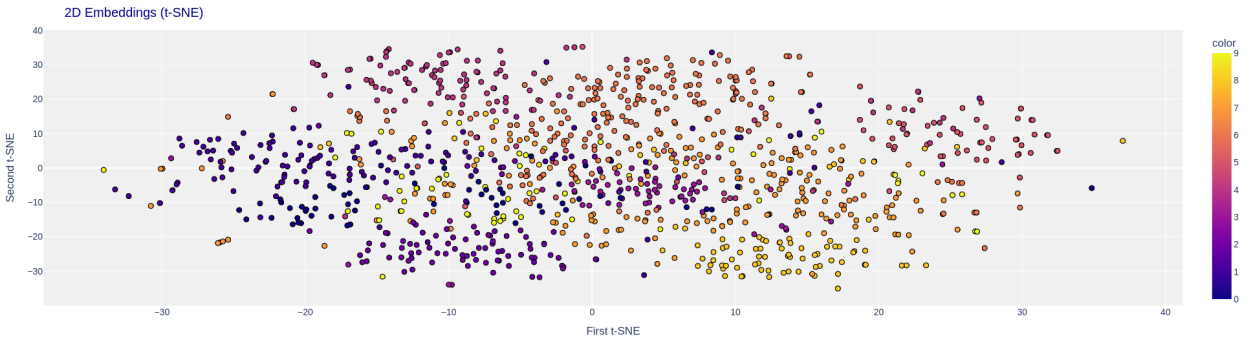

Chart 2 - t-SNE

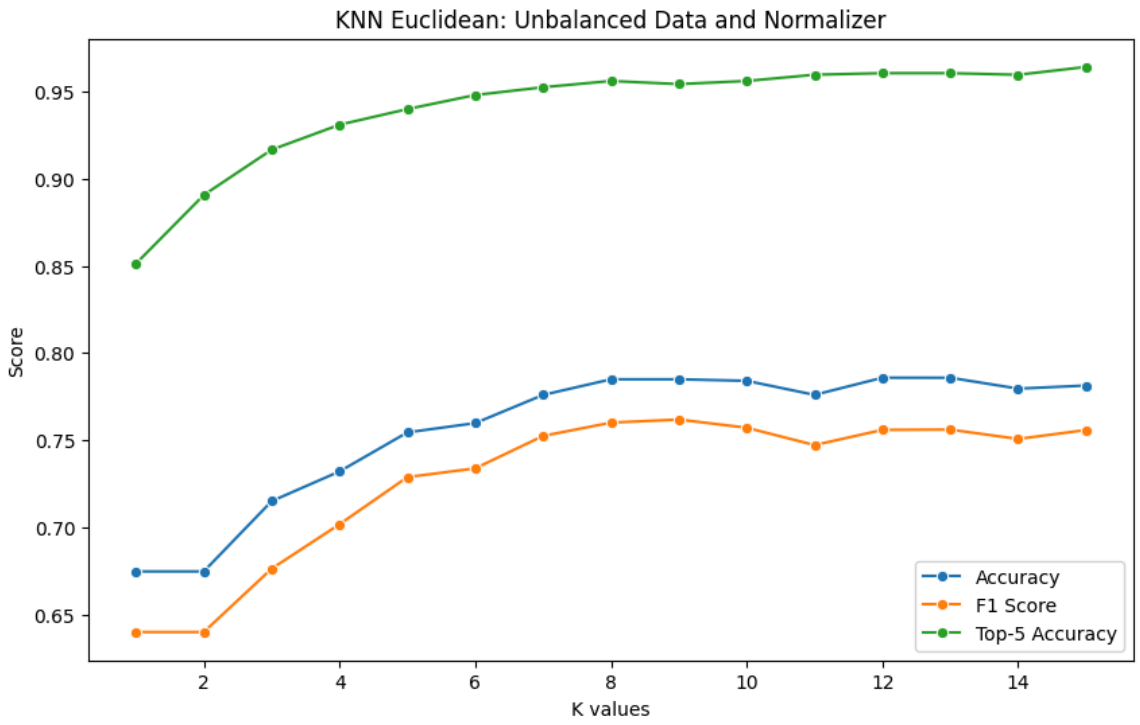

Chart 3 - KNN Euclidean using Unbalanced Data and Normalizer

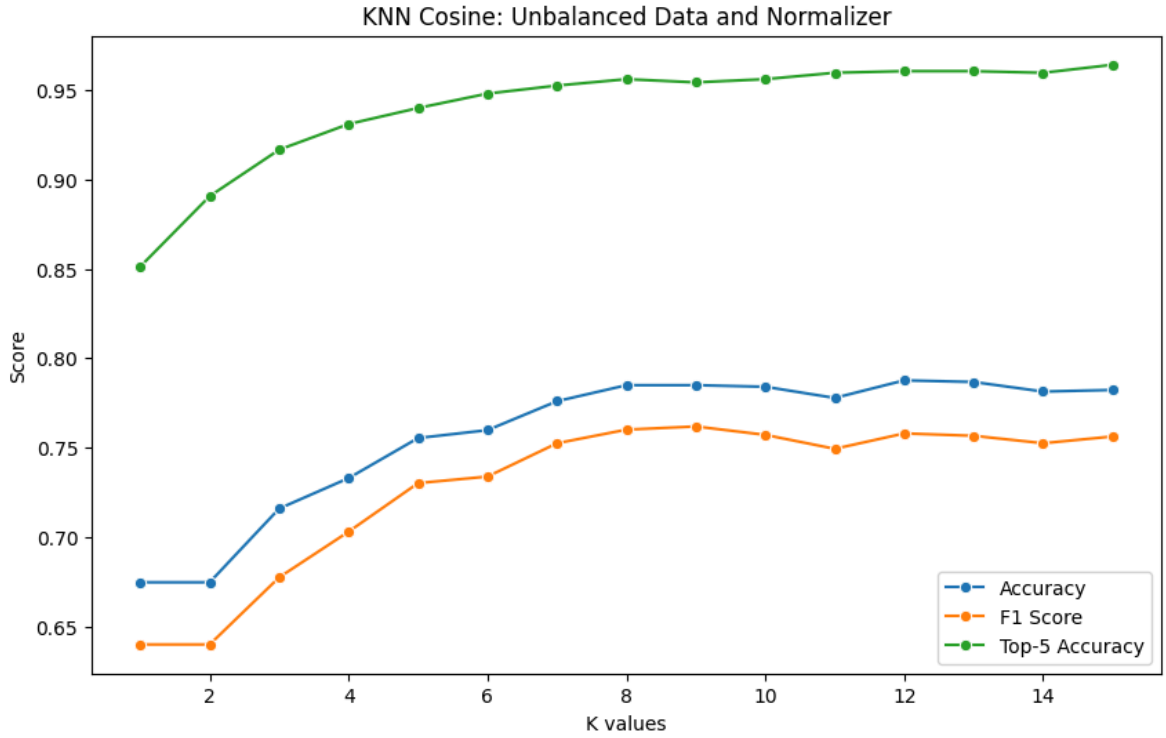

Chart 4 - KNN Cosine using Unbalanced Data and Normalizer

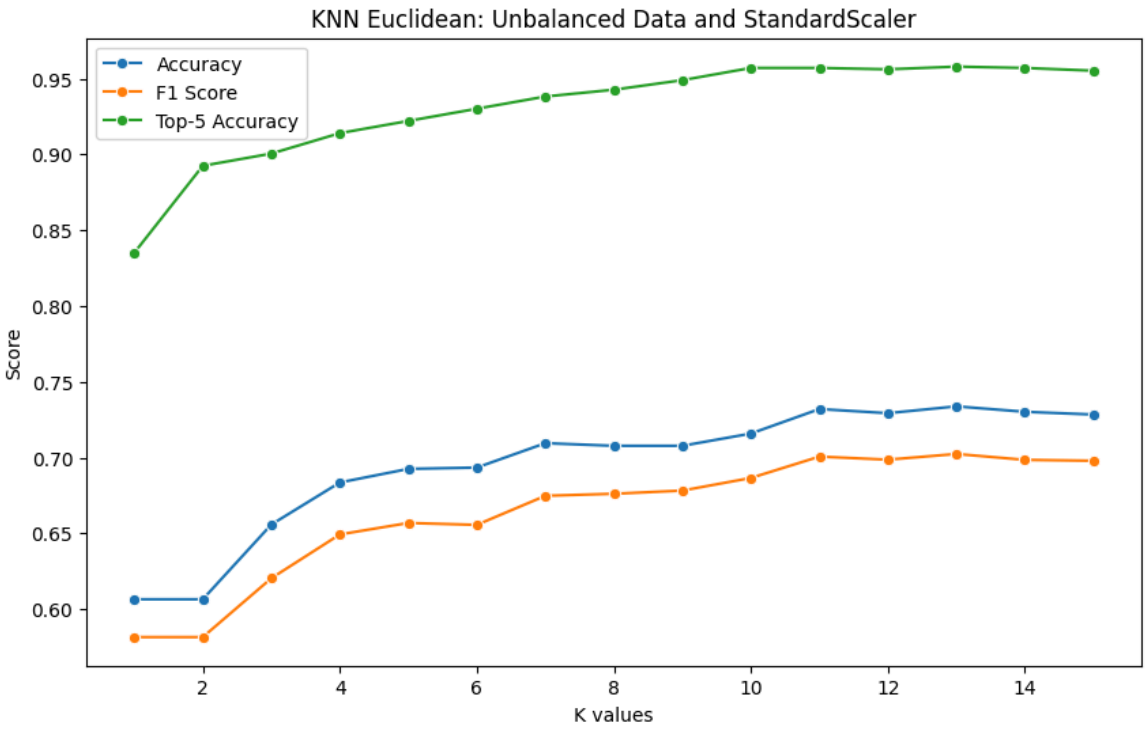

Chart 5 - KNN Euclidean using Unbalanced Data and StandardScaler

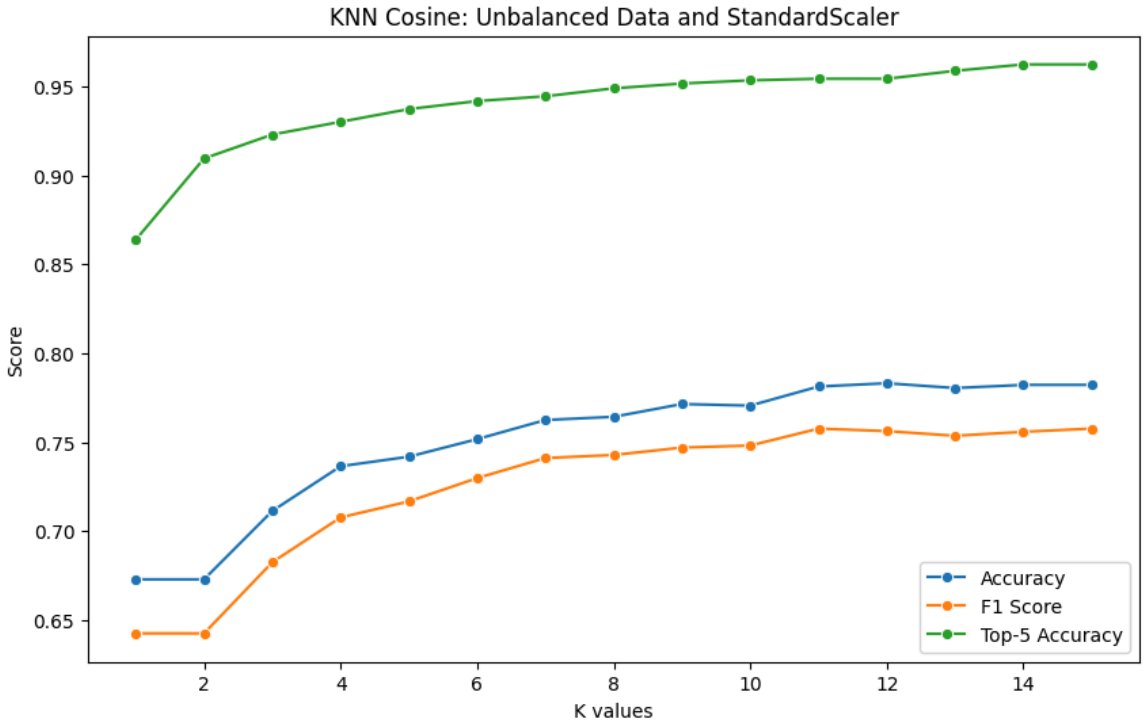

Chart 6 - KNN Cosine using Unbalanced Data and StandardScaler

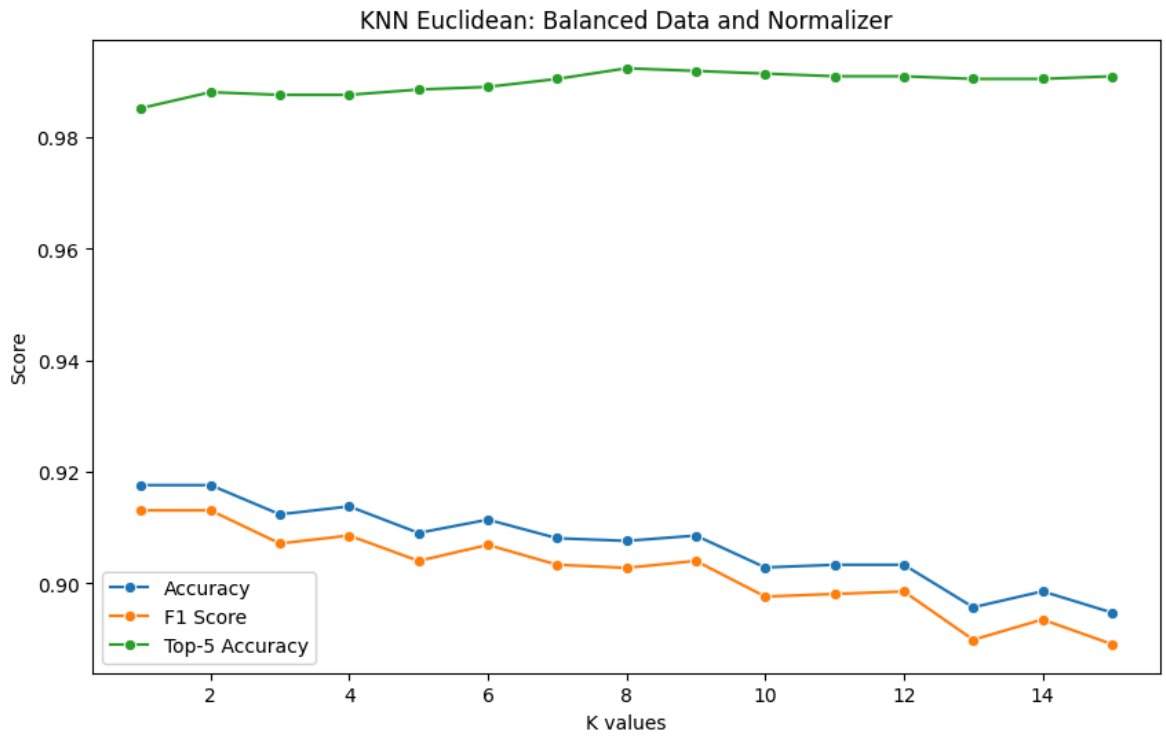

Chart 7 - KNN Euclidean using Balanced Data and Normalizer

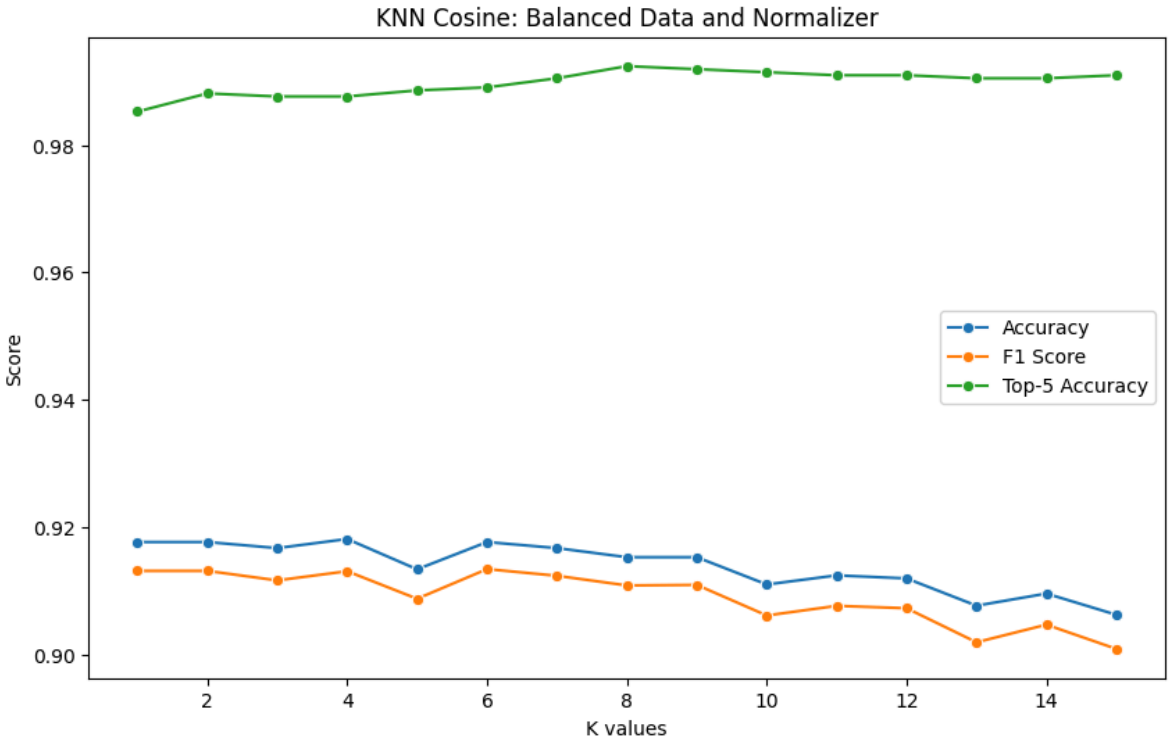

Chart 8 - KNN Cosine using Balanced Data and Normalizer

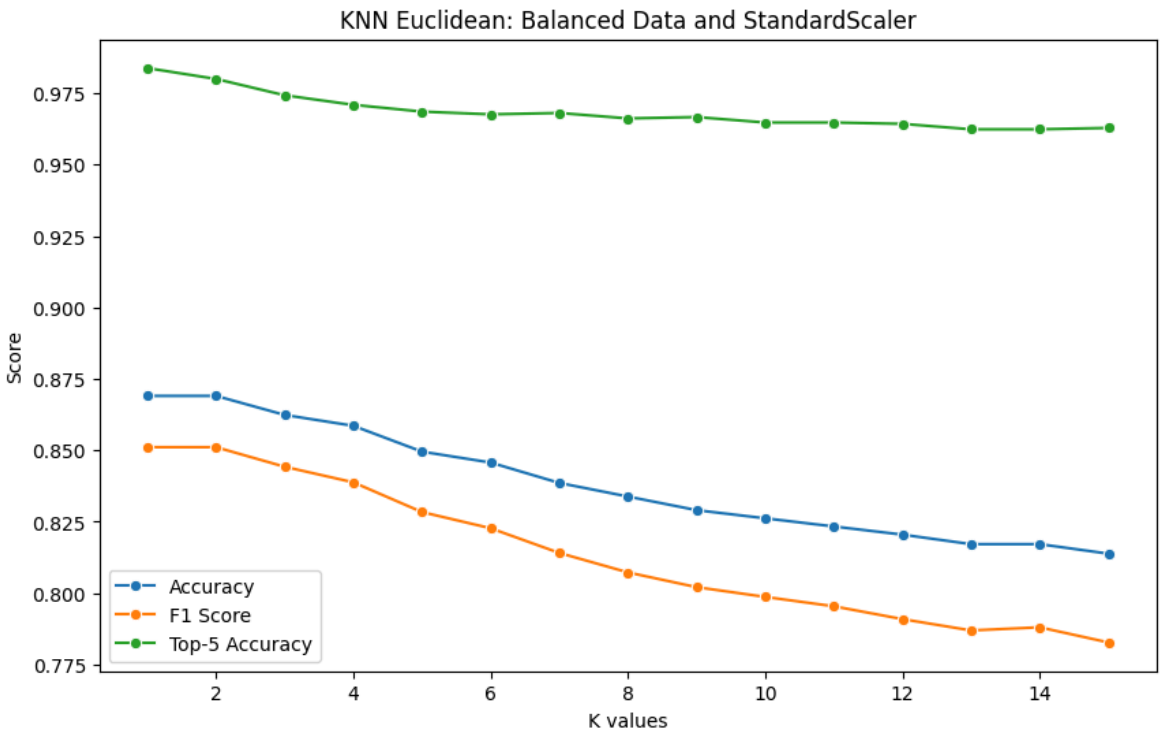

Chart 9 - KNN Euclidean using Balanced Data and StandardScaler

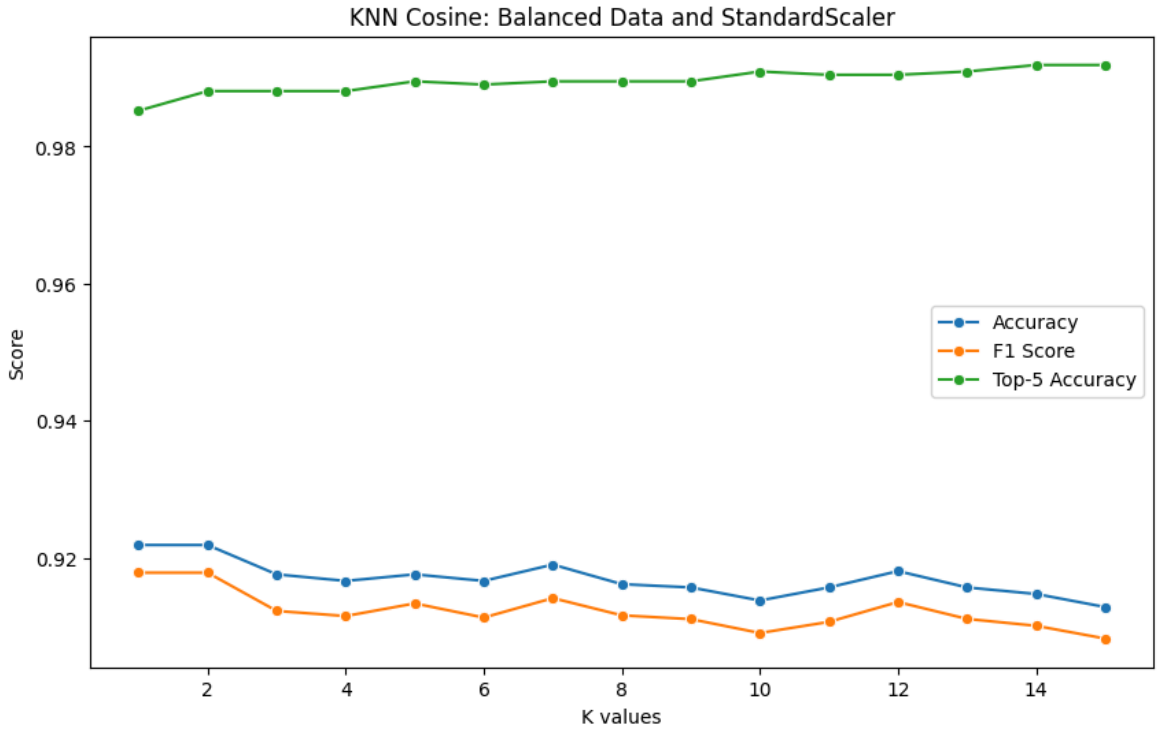

Chart 10 - KNN Cosine using Balanced Data and StandardScaler

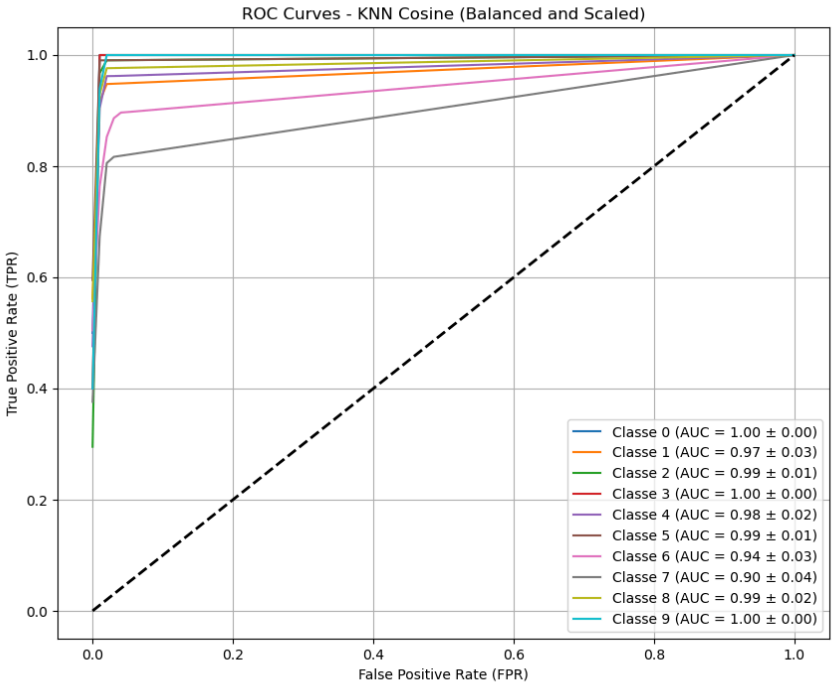

Chart 11 - ROC Curves for KNN Cosine

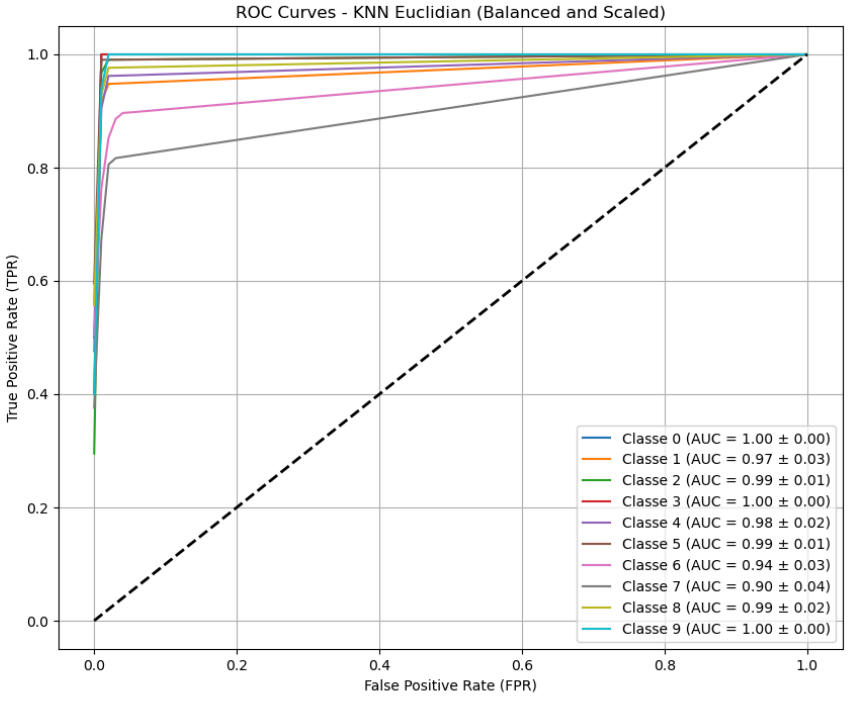

Chart 12 - ROC Curves for KNN Euclidean

3. Analysis

About the exploratory data analysis:
- All the embeddings had the same size (320).
- There were no NaN values.
- The data types were objects.
- There are 10 types of syndrome and 1116 images.

syndrome_id	Number of images
300000034	210
300000080	198
100192430	136
300000007	115
300000082	98
100610443	89
300000018	74
100180860	67
100610883	65
700018215	64

About the t-SNE:
The plot reveals a gradual transition in the color gradient, suggesting that some classes or features may have overlapping characteristics or continuous transitions. The lack of clear, isolated clusters suggests potential class overlap or mixed feature distributions in the original data. However, there are subtle groupings and localized regions where points of similar colors cluster, indicating areas of higher similarity. This result could reflect the inherent structure of the dataset and may warrant further investigation into feature importance or class boundaries.

About the classification model:
- The best model (Chart 10) was trained on a balanced dataset (using SMOTE), using 10-fold cross-validation, KNN with 2 nearest neighbors, weights='distance', metric='cosine' and scaler=StandardScaler().
- The models trained on balanced data had better performance, even though oversampling could cause overfitting and noise amplification.

- Both metrics might show the best performance at small number of nearest neighbors (like k=1 and k=2) due to the combination of normalized data and balancing, which ensures class proximity is meaningful and consistent.

- Cosine similarity is inherently affected by the normalization of feature vectors, as it measures the angle between them. Euclidean distance, when applied on normalized data, is effectively similar to comparing cosine distances for small dimensions since normalized data has unit length. As a result, normalizing the data might contribute to the performance of the two methods looking nearly identical.

About ROC Curves:
The ROC curves for both KNN Cosine and KNN Euclidean, shown in charts 11 and 12, demonstrate highly similar performance across all classes when using balanced and normalized data. Each class exhibits comparable AUC values, with most exceeding 0.85, indicating strong separability between positive and negative instances for the respective classes.
The highest AUC values are observed for Classes 2 and 4 (0.95 ± 0.04), suggesting excellent performance in distinguishing these classes. Conversely, Class 9 exhibits the lowest AUC (0.77 ± 0.09), indicating comparatively weaker performance.
The nearly identical behavior between the two distance metrics can be attributed to the normalization of the data, which minimizes differences in how the two metrics evaluate distances, and the balancing process, which ensures equitable class representation, leading to consistent results across both methods.

4. Challenges and Solutions

The hardest challenge for me was optimizing this code. As I had to test many possibilities, changing many values and methods, it would be computationally inefficient to run the models one by one. So I created dictionaries containing different types of metrics, scalers and datasets, to iterate each of them inside a triple for loop.
I also find it difficult to understand the ROC curve because it is a multiclass problem, so I had to adapt the code to work on this problem.

5. Recommendations

- Try other methods to balance the data.
- Test different fold values in cross-validation.
- Test different classification models.