Apollo Solutions Machine Learning Developer Report

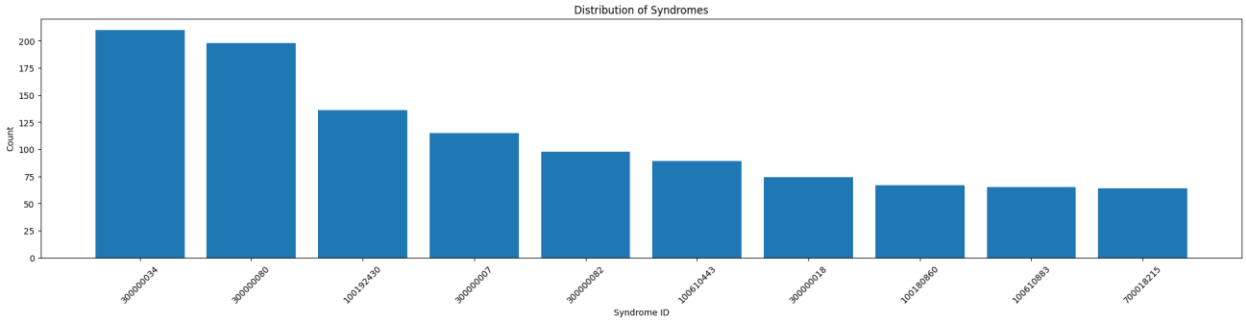
Methodology

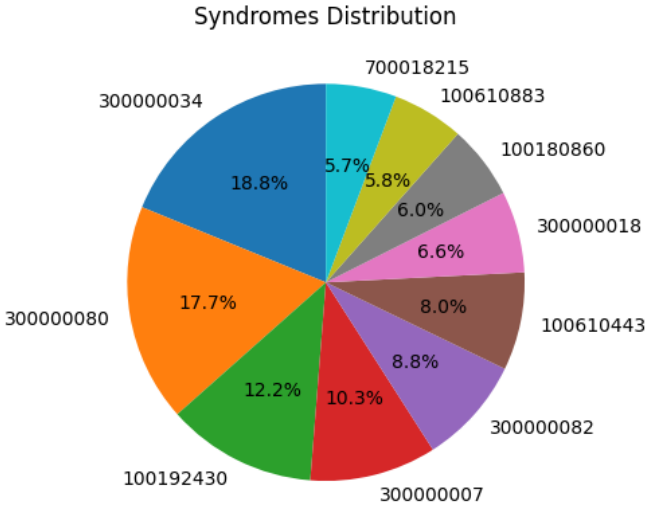
First of all, I started with the EDA of the base where I saw that the len of the variables:

```
syndrome_id      10
subject_id      941
image_id       1116
```

And the distributions of the target variable:

```
Feature: syndrome_id
syndrome_id
300000034    210
300000080    198
100192430    136
300000007    115
300000082     98
100610443     89
300000018     74
100180860     67
100610883     65
700018215     64
Name: count, dtype: int64
```



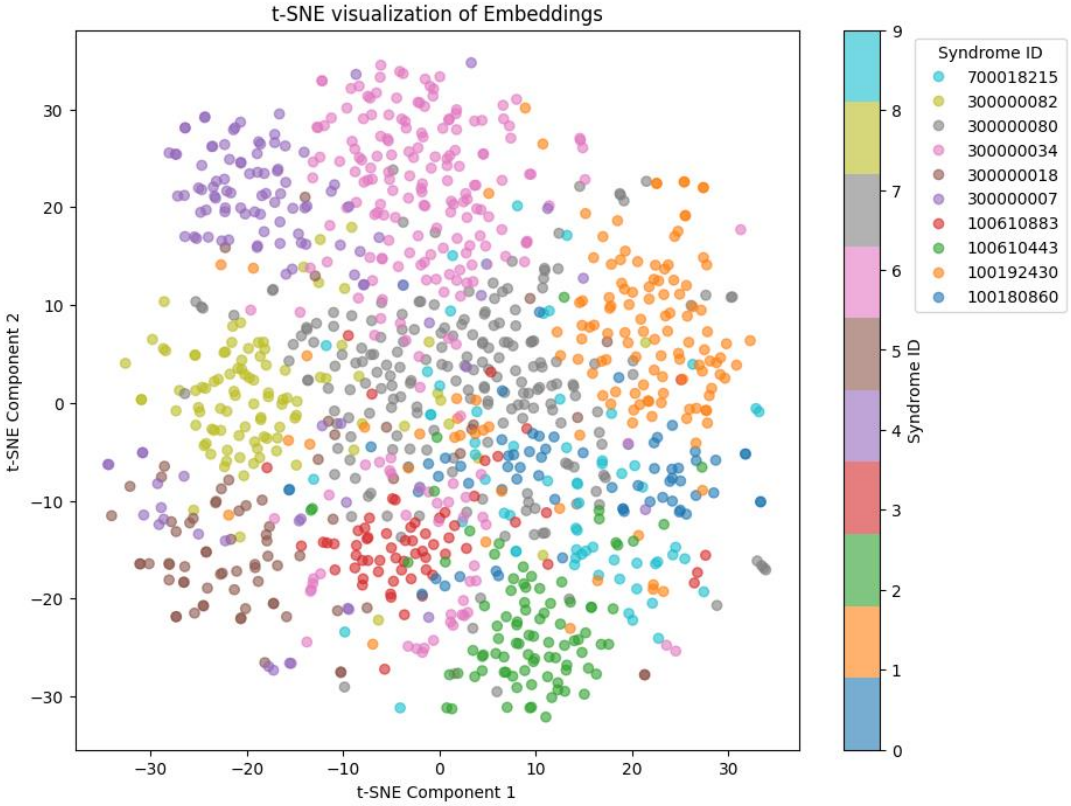


The results indicate that the dataset is imbalanced, where some classes have more representation than others it may impact the model.
Imbalanced data can lead the model to predict classes with higher representation more frequently.

We can see if this will affect our predictions and if so, we will implement augmentation methods.

Normalization of the embeddings and t-SNE:

	Mean	Std
count	323.000000	323.000000
mean	0.017023	1.273423
std	0.446074	1.159069
min	-1.207172	0.961862
25%	-0.242507	1.120811
50%	-0.013321	1.167837
75%	0.233324	1.226491
max	4.789427	16.048050

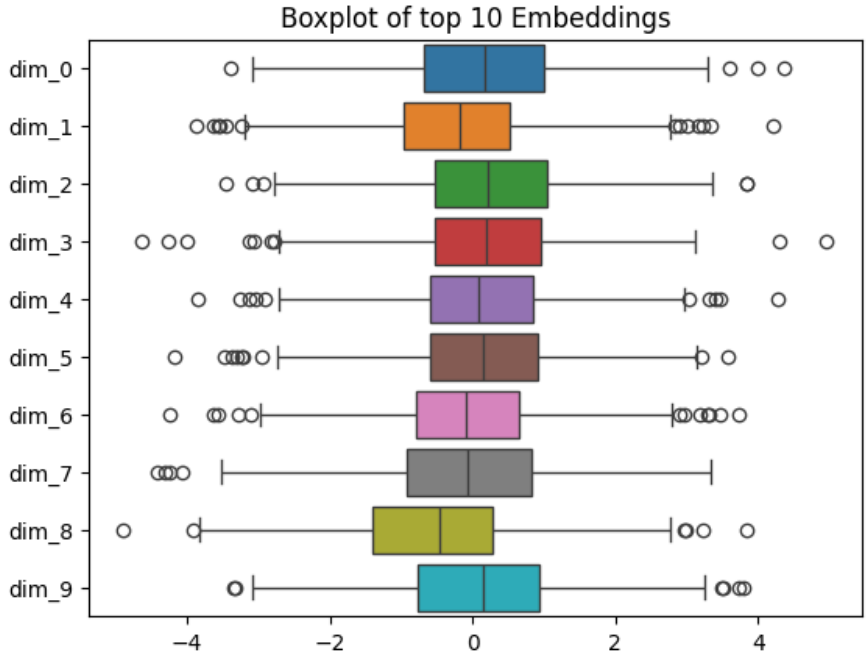


The results indicate that the embeddings are already normalized, also we can observe that the data set have some extreme values, which may indicate outliers.

In the t-SNE we can see how the syndromes are distributed, we can see that the data already contain natural clusters indicating that the data is distributed and is good for classification tasks.

These clusters indicate that each syndrome tend to form distinct groups, indicating that the embeddings contain meaningful patterns and can differentiate the classes. It is a good signal for our KNN model because it uses distance-based to classify the data.

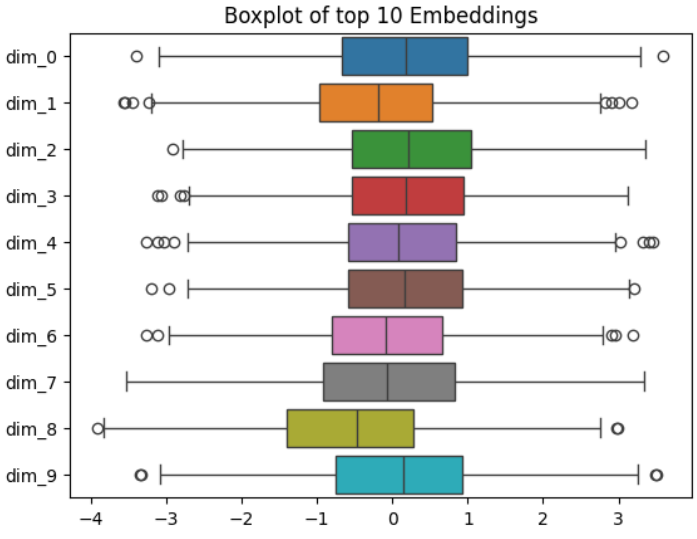
Outliers:



The results show that there are some outliers in the dataset, which can bias our Knn model since it works on distance between points.

To handle this, I decided to work with Z-score method with threshold of 3 times the standard deviation. Then I replaced the outliers with the percentile method to avoid introducing variance in the model.

```python
def replace_outliers(self, df, numeric_cols, threshold=3):
    """
    Removes outliers using the z-score method and replaces them with fixed percentiles.
    """

    if not self.predict:
        # Compute mean and standard deviation during training
        mean = df[numeric_cols].mean().astype(np.float32)
        std = df[numeric_cols].std().astype(np.float32)

        p99 = np.percentile(df[numeric_cols], 99, axis=0).astype(np.float32)
        p1 = np.percentile(df[numeric_cols], 1, axis=0).astype(np.float32)

        self.save_thresholds(p1, p99, mean, std)
    else:
        if self.thresholds is None:
            raise ValueError("Thresholds not loaded. Train the model first.")
        p1 = self.thresholds['p1']
        p99 = self.thresholds['p99']
        mean = self.thresholds['mean']
        std = self.thresholds['std']


    z_scores = (df[numeric_cols] - mean) / std

    # Replace outliers with stored percentiles
    df[numeric_cols] = np.where(z_scores > threshold, p99,
                                np.where(z_scores < -threshold, p1, df[numeric_cols]))

    return df
```



Target and Features:

```
# Defining my features and target
X = df_final.iloc[:, 3:].values
X = pd.DataFrame(X, index=df_final.index)

y = df_final['syndrome_id'].astype('category').cat.codes
y = y.copy()
```

To fit the model, I separated my data into the X (features) and y(target) variables

Augmentation:

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
```

I decided to work with an Over Sampling technique to deal with minority classes.

I will test whether this impacts the model or not.

Model overview:

The KNNClassifier class finds the best k for classification using cross-validation and then trains the final model with the best k.

Steps:

1. Tests multiple k values (1 to 15) using cross-validation.
2. Evaluates performance using AUC, F1-score, Accuracy, and Top-K Accuracy.
3. Selects the best k based on AUC.
4. Trains the final KNN model with the best k.
5. Provides predictions and probability.

Results And Analysis

```
models = {
    "knn_model_cos_resampled": KNNClassifier(X_resampled, y_resampled, (1, 15), "cosine"),
    "knn_model_cos": KNNClassifier(X, y, (1, 15), "cosine"),
    "knn_model_euc_resampled": KNNClassifier(X_resampled, y_resampled, (1, 15), "euclidean"),
    "knn_model_euc": KNNClassifier(X, y, (1, 15), "euclidean")
}
```

I decided to train 2 models for the 2 metrics.

The model with _resampled is trained with the resample data set that handles imbalanced data, and the other one is trained with the normal data set. With that we can see if the Smote method can improve our model

	Model Name	k	Distance Metric	AUC	F1-Score	Accuracy	Top-K Accuracy
0	knn_model_cos_resampled	15	cosine	0.989426	0.863582	0.867143	0.947619
1	knn_model_cos	15	cosine	0.965121	0.785351	0.790347	0.945946
2	knn_model_euc_resampled	14	euclidean	0.970600	0.755916	0.781429	0.909524
3	knn_model_euc	15	euclidean	0.952049	0.746005	0.754344	0.900901

Best Model: knn_model_cos_resampled | k=15 | AUC=0.9894 | Distance: cosine

In the results, we can see thar our best model is k=15, metric=cosine and using the resampled data set.

The model trained with the resampled data set show a good improvement in the metric, specifically into the F1-score.

Metric : Cosine

	precision	recall	f1-score	support
0	0.79	0.78	0.78	67
1	0.85	0.86	0.86	136
2	0.89	0.96	0.92	89
3	0.84	0.65	0.73	65
4	0.82	0.83	0.82	115
5	0.80	0.72	0.76	74
6	0.84	0.96	0.90	210
7	0.81	0.84	0.82	198
8	0.82	0.85	0.84	93
9	0.89	0.53	0.67	64
accuracy			0.83	1111
macro avg	0.84	0.80	0.81	1111
weighted avg	0.83	0.83	0.83	1111

Metric : Cosine using SMOTE

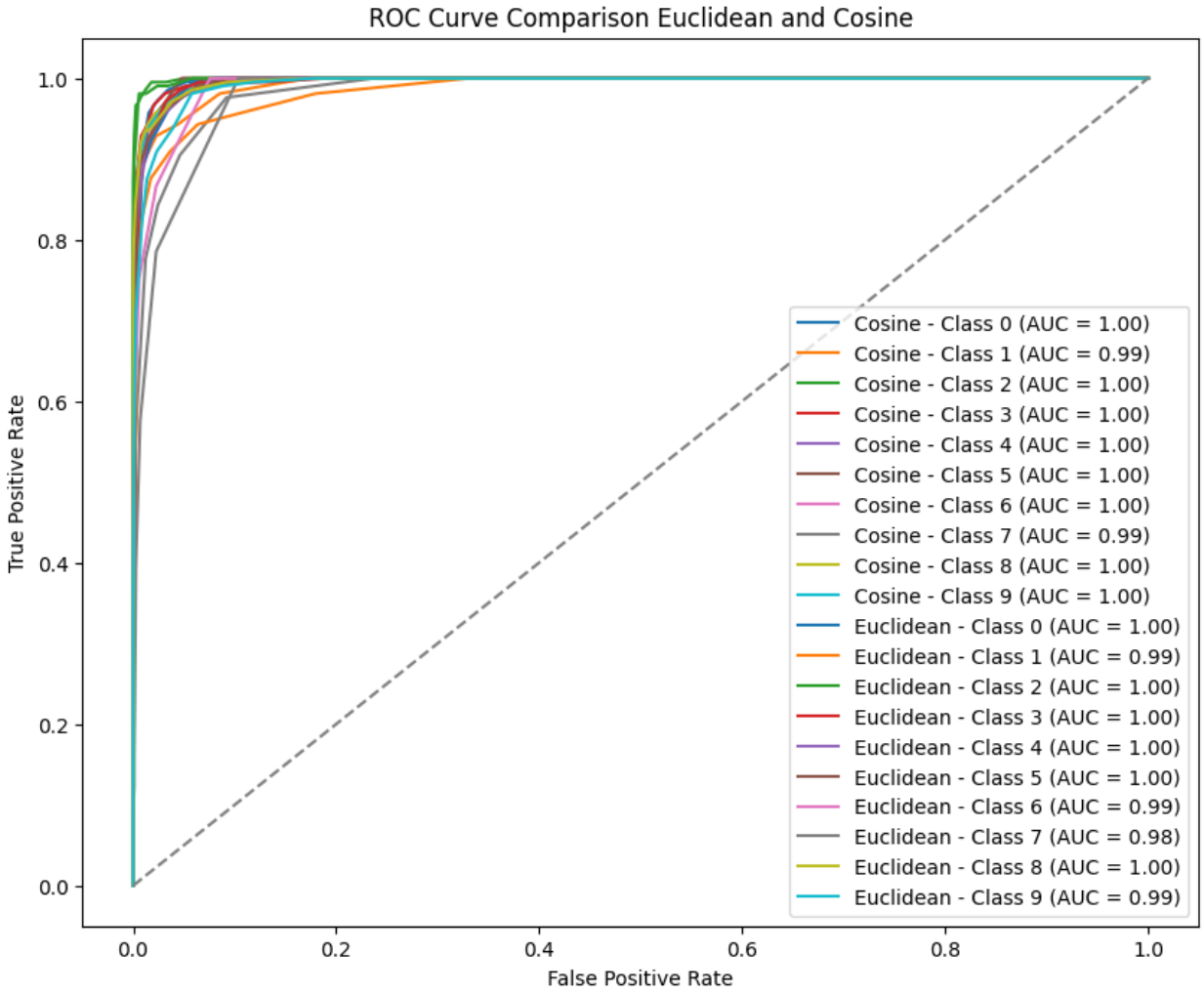
	precision	recall	f1-score	support
0	0.87	0.96	0.91	210
1	0.92	0.89	0.90	210
2	0.93	0.98	0.96	210
3	0.88	0.95	0.91	210
4	0.87	0.92	0.90	210
5	0.90	0.91	0.91	210
6	0.96	0.84	0.90	210
7	0.95	0.62	0.75	210
8	0.84	0.96	0.90	210
9	0.87	0.92	0.89	210
accuracy			0.89	2100
macro avg	0.90	0.89	0.89	2100
weighted avg	0.90	0.89	0.89	2100

In the results we can see that the class 9 was impacting the model performance, when we balanced this data the model has a good improvent in performance specifically into the imbalanced data.

The Cosine metric performed better because it tends to work better with high dimensional data and with non-linear relationships that is the case of images and our embeddings. On the other hand, the Euclidean metric is more suitable for data that follows a linear distribution

In the graph bellow we can see the Auc-Roc results between Euclidean and Cosine metrics.

Even though the 2 metrics have similar results [Cosine:0.989 , Euclidean:0.970], the cosine is better in the other metrics and also a little bit better in Auc metric.



Challenges and Solutions

Data Imbalance: The primary challenge was the syndrome class imbalance in the dataset, which can lead the model to predict the more frequent classes.

To handle this, I used an over sampling technique (SMOTE) to balance the dataset by generating more samples for the minority classes.

Outliers in Data: The embeddings contained extreme values that may potentially bias the model since it is distance based.

To handle this, I used the Z-score method with a threshold of 3 standard deviations. I replaced them using the percentile method to prevent introducing additional variance into the model.

Distribution and Normalization in Data: The embeddings were provided without explanation, so I checked the mean and std to determine if the data needed normalization. If the mean was far from zero or there was high variance, I need to normalize the data. But in this case the data already has a good normalization.

I also used t-SNE to reduce the dimensionality of the embeddings to 2D. This helped me visually check if there were natural clusters in the data, which in our case where the data is well distributed and has natural clusters

Recommendations

Augmentation Techniques: We can add additional augmentation techniques, such as rotations or transformations of images, to generate more diverse training samples.

Hyperparameter Tuning for KNN: Although I used cross-validation to determine the optimal value of k (15). Further hyperparameter tuning may can improve the model performance.

Explore More Complex Models: While KNN had a good performance, we can achieve even a better performance by using more advanced models such as SVM, Random Forest, or neural network-based.