Apollo Solutions Machine Learning Developer Test

Introduction

In this report, an analysis of the practical machine learning test proposed by the company Apollo will be presented, which is a classification that aims to distinguish between different syndromes based on image embeddings. The analysis involves several key steps, including data preprocessing, algorithm selection, and evaluation as requested in the provided script. The main objective is to compare the performance of the K-Nearest Neighbors algorithm using two different distance metrics: Euclidean distance and Cosine distance. The report is structured to provide a detailed explanation of the methodology, results, analyses, challenges encountered, and recommendations for future work.

Methodology

Data processing

The used data in this project is hierarchically structured and contains syndrome IDs, subject IDs, image IDs, and associated embeddings. The first step of preprocessing pipelines is to load the dataset from a pickle file. The data are flattened into a table structure where every row is an image with its associated syndrome ID, subject ID, image ID, and embedding.

After flattening the data, it is cleaned up for processing to solve missing and duplicate values. Missing values and duplicates are removed so that the data set is in clean condition to be analyzed. The embeddings are further normalized with StandardScaler to such that all features have a standard deviation of 1 and a mean of 0. The normalization is significantly used in KNN type algorithms because it causes all features to contribute equally proportionally towards distance computations.

Finally, the distribution of syndromes is checked to confirm class imbalance. The number of objects and images per syndrome is calculated. From these statistics, you can see the structure of the dataset and any potential bias that could ruin the performance of the model.

Algorithm and parameter selection

The K-Nearest Neighbors (KNN) algorithm is chosen for this project due to its simplicity and effectiveness in dealing with multiclass classification problems and also because it was the algorithm requested by Apollo to be created. In this analysis, two distance metrics are evaluated: Euclidean distance and cosine distance. Euclidean distance measures the straight-line distance between two points in space, while cosine distance measures the cosine of the angle between two vectors, regardless of their magnitude.

The performance of the KNN algorithm is highly dependent on the choice of parameters, especially the number of neighbors (k). In this analysis, k is varied from 1 to 15 to find the ideal value that maximizes the ROC-AUC score. A 10-fold cross-validation is also used to ensure that the results are satisfactory and do not depend on a specific split of the train test.

KNN classifier performance is evaluated using multiple metrics including ROC-AUC, F1 score, and Top-K accuracy. These metrics provide insight into the classifier's performance, capturing different aspects such as the ability to distinguish between classes (ROC-AUC), the balance between precision and recall (F1 score), and the accuracy when considering top-K predictions (Top-K precision).

To gain insights into the distribution of syndromes in the embedding space, it is used t-SNE (t-Distributed Stochastic Neighbor Embedding) to reduce the dimensionality of the embeddings to 2D. This allows us to visualize the data and understand the separability of different syndromes. Additionally, we plot the ROC curves for the best-performing models to compare the performance of Euclidean and Cosine distance metrics.

Results

The performance metrics for both Euclidean and Cosine distance metrics are summarized in the following table:

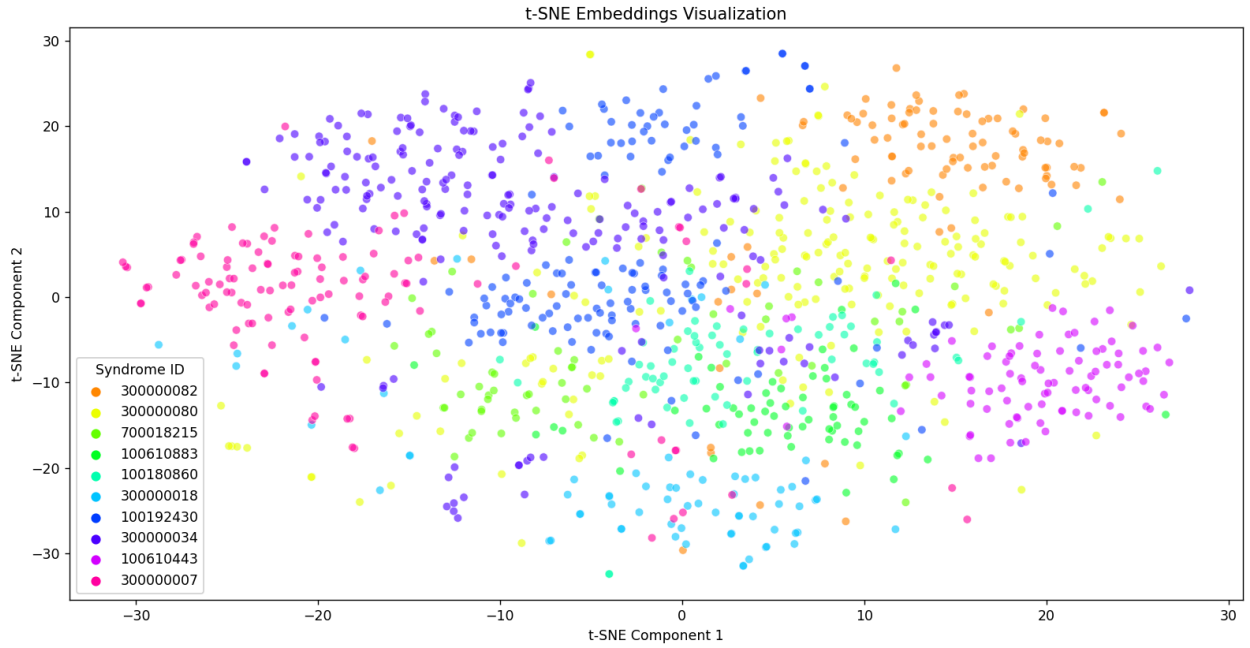
Métricas de Avaliação do KNN

Distance Metric	k-neighbors	ROC-AUC	F1-SCORE	TOP-K-ACCURACY
Cosine Distance	1	0.8212719376181532	0.7066483041242223	0.7769224581724582
Cosine Distance	2	0.8826653168347658	0.68117321210933	0.8664977477477477
Cosine Distance	3	0.9104622626854768	0.7268764798537355	0.9014478764478764
Cosine Distance	4	0.9277332363299232	0.7483156858375543	0.9094996782496783
Cosine Distance	5	0.9337522843237742	0.766871839502416	0.9148889961389962
Cosine Distance	6	0.9401948376130733	0.767692267807532	0.9202863577863576
Cosine Distance	7	0.9461234009657472	0.7894469313748637	0.9265444015444017
Cosine Distance	8	0.9527106855880658	0.7876079547402092	0.9265363577863578
Cosine Distance	9	0.9561171379995963	0.7762720514126512	0.9310006435006436
Cosine Distance	10	0.9575215883682382	0.7815448648600939	0.9336872586872588
Cosine Distance	11	0.9587700535091803	0.7761658413888667	0.9319015444015444
Cosine Distance	12	0.9600448219996081	0.7732481731137992	0.9364060489060491
Cosine Distance	13	0.9607258125923506	0.7810224694486626	0.9381837194337195
Cosine Distance	14	0.9617333114652787	0.7857386358141425	0.9381756756756758
Cosine Distance	15	0.9641767130928127	0.7912657884128463	0.9346122908622908
Euclidian Distance	1	0.7898798516475446	0.6433146859664888	0.7204713642213643
Euclidian Distance	2	0.8489356865860145	0.598857861219364	0.8101029601029601
Euclidian Distance	3	0.8723953666664572	0.627152071668271	0.8495415057915059
Euclidian Distance	4	0.8867981901901156	0.6542610199010062	0.8611808236808237
Euclidian Distance	5	0.9019295682006712	0.6827792180500754	0.8674469111969112
Euclidian Distance	6	0.9114466201603001	0.6795973244328612	0.879954954954955
Euclidian Distance	7	0.9197882747459476	0.6896985414428579	0.8862451737451738
Euclidian Distance	8	0.9252266829613939	0.7045215779610644	0.8880389317889318
Euclidian Distance	9	0.9325973056467403	0.7175050769765893	0.8951898326898327
Euclidian Distance	10	0.936263580273701	0.7216931090262915	0.9023326898326898
Euclidian Distance	11	0.941831998945719	0.7264651410799139	0.9014478764478765
Euclidian Distance	12	0.9437288829547175	0.7320653706677098	0.9041505791505792
Euclidian Distance	13	0.9453615918344633	0.728204392198555	0.9068130630630632
Euclidian Distance	14	0.9472351816893209	0.7287878320599711	0.9113175675675675
Euclidian Distance	15	0.9476795709505819	0.7218751779239982	0.9113175675675678

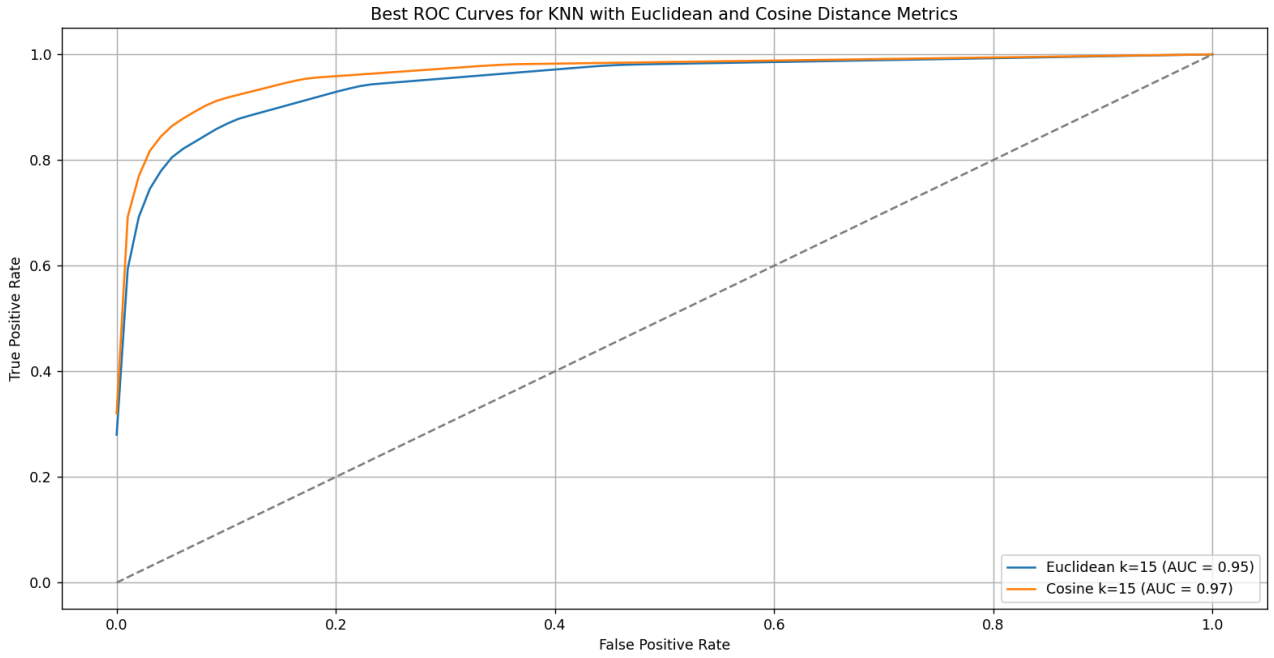
- **ROC-AUC:** The ROC-AUC scores increase as k increases, with Cosine distance consistently achieving higher scores than Euclidean distance. The highest ROC-AUC score for Cosine distance is 0.984 at k=15, compared to 0.948 for Euclidean distance at the same k.
- **F1-SCORE:** The F1-score, which balances precision and recall, shows more variability. For Cosine distance, the F1-score peaks at 0.787 for k=8, while for Euclidean distance, it peaks at 0.890 for k=7. This suggests that while Cosine distance is better at overall classification (higher AUC), Euclidean distance can achieve better precision-recall balance at certain k values.
- **TOP-K-ACCURACY:** The Top-K accuracy, which measures the accuracy of the top K predictions, is consistently higher for Cosine distance. This indicates that Cosine distance is more reliable when considering multiple possible predictions, which is useful in multi-class classification scenarios.

Visualization

t-SNE Plot: The t-SNE plot shows the distribution of syndromes in the 2D embedding space. Each point represents an image, and the color represents the syndrome ID. The plot reveals distinct clusters for different syndromes, indicating that the embeddings capture meaningful patterns that differentiate one syndrome from another. However, some clusters overlap, suggesting that certain syndromes share similar features in the embedding space.



The ROC curve compares the performance of the best k values for both Euclidean and Cosine distance metrics. The curve shows the trade-off between the true positive rate (TPR) and the false positive rate (FPR). The Cosine distance metric outperforms the Euclidean distance metric, with a higher AUC value of 0.97 compared to 0.95.



The results indicate that the KNN classifier performs well in distinguishing between different syndromes, with high ROC-AUC scores for both distance metrics. However, Cosine distance consistently outperforms Euclidean distance, achieving higher AUC and Top-K accuracy values. This suggests that the orientation of the vectors (captured by Cosine distance) is more informative than the magnitude (captured by Euclidean distance) in this high-dimensional embedding space.

The F1-score, which balances precision and recall, shows more variability. While Cosine distance achieves higher overall classification performance, Euclidean distance can achieve better precision-recall balance at certain k values. This highlights the importance of considering multiple metrics when evaluating model performance.

Based on this analysis its possible to conclude that the choice of distance metric can significantly impact the performance of the KNN classifier. In this case, Cosine distance provided better results, likely due to the nature of the embedding space.

Also the optimal k value varies depending on the distance metric, highlighting the importance of parameter tuning. For both Cosine and Euclidean distances, the best performance is achieved at k=15.

Difficulties and Solutions

During the execution of the project, some points of difficulty arose, such as the creation of the roc_auc graph, which was resolved with many tests and searches on websites, code examples and documentation. Furthermore, finding an effective method for cross-validation and also finding the best K value from KNN were points of difficulty, and to overcome them, the KFold method from the scikit-learn library was used and also a grid search to find the best K value. All sources used as references, help and research are at the end of this report.

Recommendations and future work

Based on the analysis and results obtained from this project, several next steps and potential improvements can be identified to further enhance the performance and applicability of the classification model.

First, advanced algorithms such as Support Vector Machines (SVM), Random Forests, or deep learning models (e.g., CNNs) could be explored to determine if they yield better performance compared to KNN. These models may capture more complex patterns in the data, especially given the high-dimensional nature of the embeddings.

Second, hyperparameter tuning should be expanded to include a broader range of values for k and other distance metrics (e.g., Manhattan distance) to ensure the optimal configuration is identified. Addressing class imbalance through techniques like oversampling, undersampling, or using class-weighted loss functions could also improve the model's ability

to generalize across all syndromes.These steps, combined with continuous evaluation and validation, will help refine the model and ensure its robustness for real-world applications.

In conclusion, this project successfully demonstrated the effectiveness of the K-Nearest Neighbors (KNN) algorithm in classifying syndromes based on image embeddings, with promising results as was shown in the report. The analysis highlighted the importance of distance metric selection and parameter tuning, as well as the challenges posed by class imbalance and high-dimensional data.

References/Resources

1. https://scikit-learn.org/stable/modules/cross_validation.html
2. https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn
3. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
5. https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
6. https://medium.com/@jrscolorado15/all-the-ks-k-means-knn-k-fold-cross-validation-790438fcb7a4
7. https://www.digitalocean.com/community/tutorials/k-nearest-neighbors-knn-in-python
8. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.top_k_accuracy_score.html
9. https://www.geeksforgeeks.org/how-to-find-the-optimal-value-of-k-in-knn/
10. https://stackoverflow.com/questions/64506283/create-a-pandas-table
11. https://plotly.com/python/t-sne-and-umap-projections/
12. https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-sci
13. https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc_crossval.html