Predicting Genetic Disorders: A Machine Learning Approach

Salma Loukili, Nicolas Gutierrez

Abstract

This study attempts to predict genetic disorders using machine learning; focusing on Random Forest and Neural Networks. Addressing a multi-class classification challenge, a dataset rich in genetic features was processed, despite its substantial missing values and class imbalances. Through detailed pre-processing and feature engineering, the models selected yield encouraging results, with the Neural Network slightly surpassing Random Forest in key performance metrics. This research highlights the potential of computational methods in advancing precision medicine and sets the stage for further improvements in diagnostic accuracy.

December 18, 2023

I. Introduction

THIS paper address the critical challenge of predicting genetic disorders, crucial for healthcare and precision medicine. Genetic disorders, resulting from DNA mutations, lead to various diseases, highlighting the need for efficient predictive models. The goal is to enhance diagnostic accuracy and facilitate early interventions. The algorithm processes patient data diagnosed with genetic disorders. The input includes numerical and categorical features like blood tests, symptoms, genetic history, birth complications, and other disorder indicators.

The output comprises two classes: *Genetic Disorder* and its respective *Disorder Subclass*. Nine Genetic Disorder Sub-classes within three primary Genetic Disorders: *Single-gene inheritance diseases, Mitochondrial genetic inheritance disorders, and Multi-factorial genetic inheritance disorders*. These include the following subclasses: *Tay-Sachs, Hemochromatosis, Cystic Fibrosis (Single-gene), Leber's Hereditary Optic Neuropathy, Leigh Syndrome and Mitochondrial Myopathy (Mitochondrial), Diabetes, Cancer and Alzheimer's (Multifactorial).* This project tackles a multi-class classification problem to predict these disorders.

The aim is to employ machine learning techniques, focusing on Random Forest and Neural Networks, to precisely predict genetic disorders and subclasses.

II. Related Work

The main paper that will be evaluated [1] uses different methods to classify the *Genetic Disorder* and *Disorder Subclass*. The dataset used is the same [2], but the methods differ significantly. Raza et al. make two key contributions: first, a novel feature engineering method that combines class probabilities from Extra Tree (ET) and Random Forest (RF) to create a feature set for model training. Second, they employ the classifier chaining approach; where multiple classifiers are linked in a chain to make predictions. Evaluation metrics for the multi-label multi-class problem include macro accuracy, f1 score, Hamming loss, and α-evaluation score. The results indicate that Extreme Gradient Boosting (XGB) outperforms other models, achieving a 92% α-evaluation score and an 84% macro accuracy score. This shows superior performance compared to state-of-the-art methods, considering both effectiveness and computational complexity.

All the features are subsequently processed using a novel technique named ETRF. This consists in feeding the dataset to two models, an Extended Tree and a Random Forest. The class predicted probability of each is then used as input for different chained classifiers. This is a technique used in other papers by the same authors, and it is stated that "it is empirically found that the use of prediction probabilities as features for model training yields better results as compared to using raw features". [3].

Classifier chaining involves creating a connected chain of multiple classifiers for the machine learning model [4]. Each classifier in the chain predicts in the specified order, and the predictions from earlier models in the chain are considered by subsequent models. The number of classifiers in the chain corresponds to the number of classes in the dataset under investigation. Several models were used, and the best result was obtained with XGBoost [5], nevertheless the technique worked with several other classifiers comparably.

III. Dataset and Features

A. Dataset analysis

The dataset, initially sourced from Kaggle and derived from the 2021 HackEarth challenge [2], presents a complex array of features with significant missing values, as illustrated in Plot 1. The figures, and the analysis, were made using Pandas [6], NumPy [7], Seaborn [8], and Matplotlib [9]. This plot, displaying each feature as a bar colored by its unique values, reveals two prominent characteristics: the prevalence of missing values across features and the dominance of binary categorical data within the dataset.

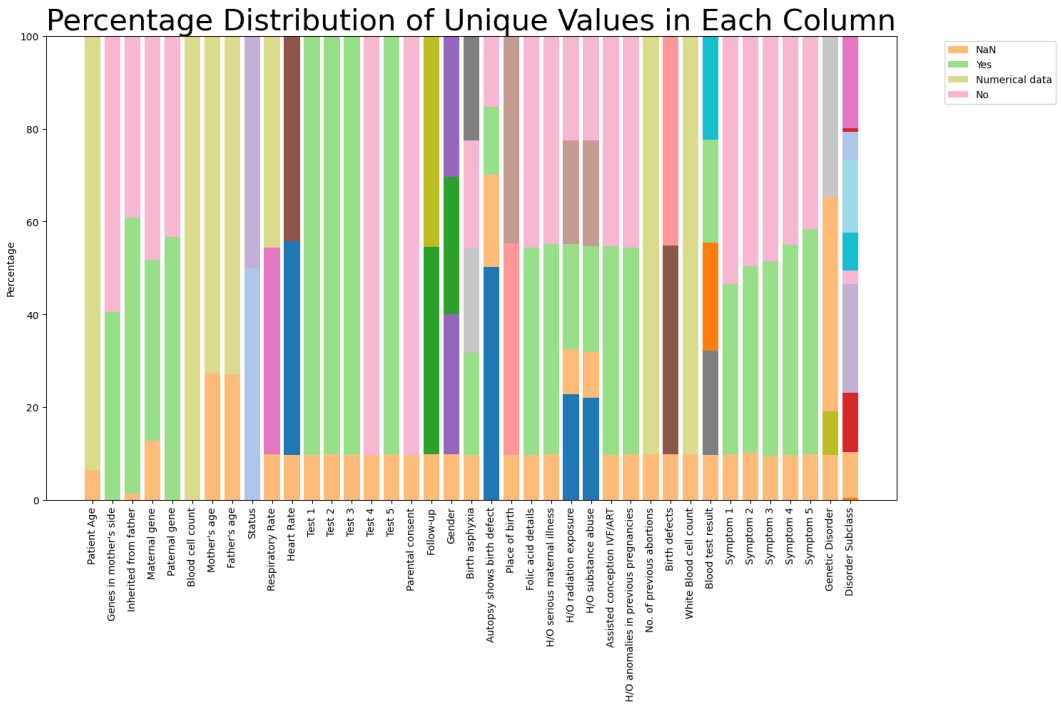
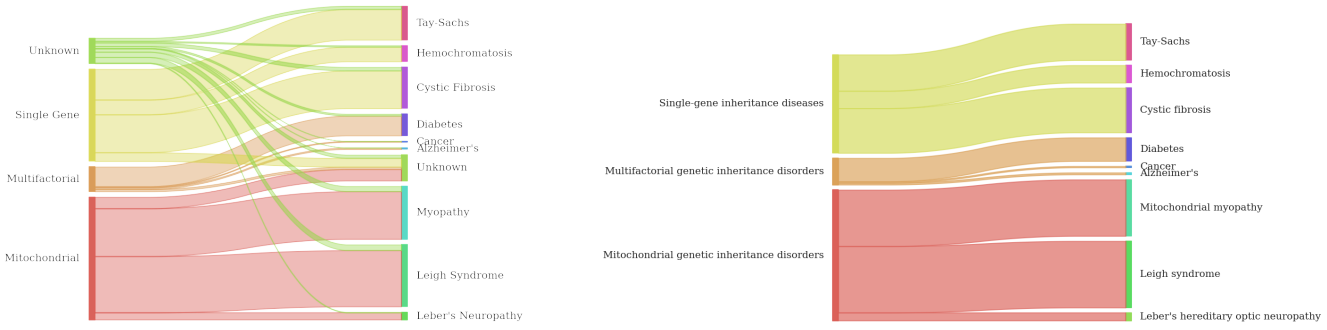


Figure 1: Overview of unique values of every feature.

Crucially, the dataset's last two features represent target outcomes. Plot 2 show the relationships between all the disorders and subclasses. This visualization helped by showing unknown values in the *Genetic Disorder* feature where the *Subclass* is known.



(a) Before connecting disorder unknowns.

(b) After connecting disorder unknowns.

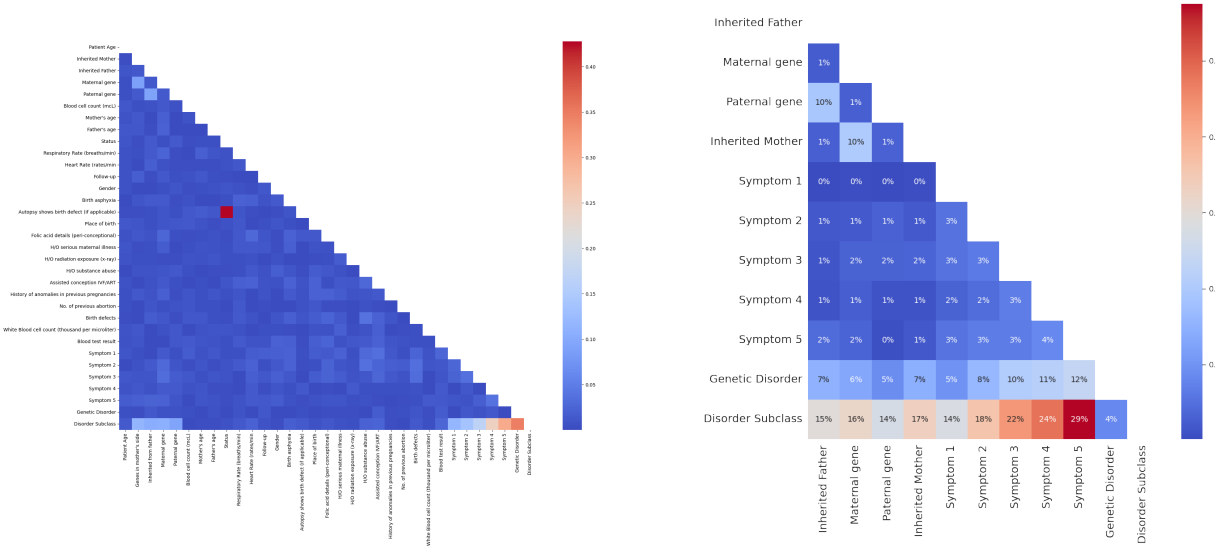
Figure 2: Sankey plot of Genetic Disorders and Disorder Subclasses.

Also portrayed in the Sankey plot 2, one of the dataset's challenges is the class imbalance, particularly in the main target class. The *Mitochondrial* and *Single-gene* largely outnumber the *Multifactorial* class. Although oversampling techniques were explored (discussed in Section IV: Methods), they did not significantly enhance accuracy. More sophisticated approaches, as discussed in Section II: Related Work, might be better suited for addressing this imbalance.

B. Pre-processing steps

The dataset initially consists of 22,083 entries, requiring multiple pre-processing steps for effective model training.

- **Feature Selection:** Given the vast number of features and the pervasive unknowns, selecting the right features is critical for model performance. Irrelevant features like *Family Name, Parental Consent, and Patient ID* were removed. The correlation heat-map in figure 3 (a) demonstrates weak feature-disease correlations, with exceptions highlighted in figure 3 (b). These selected features (b) consistently yield better results, while adding others either had no effect or reduced performance.



(a) Heatmap of all feature correlations.

(b) Heatmap of selected feature correlations.

Figure 3: Cross Correlation heatmaps of the dataset.

- **Missing Values**
  The dataset's significant proportion of missing values required careful handling. Initially, unknowns were classified as a separate category, but this approach was abandoned due to unsatisfactory results. Numerical features were also imputed using the *Median Imputer* [10], much to the same effect. For categorical features, various imputation methods were tested, but ultimately, rows with unknown labels were dropped for optimal performance.
- **Encoding & Normalizing**
  All categorical data were converted to numerical formats using label encoding. For the Neural Network, target features underwent one-hot vector encoding. Finally, all columns were then normalized using scikit-learn's *Robust Scaler* [10]. This scaler outperformed other scaling methods that were tested.

C. Dataset Split

The dataset was ultimately divided into training, testing, and validation sets. For the Neural Network, a 70/10/20 train/test/validation split was chosen. For the Random Forest, a 90/10 train-test split was used, together with 5-fold cross validation on the training data. This proved to be the highest performing split without compromising test significance. Different splits were explored throughout the project to accommodate various models.

After completing an extensive pre-processing procedure, 11660 entries and 10 features were compiled for model analysis.

IV. Methods

A. Classical ML Approaches

The first approach used to predict the *Disorder* and its *Subclass* using the selected features is Random Forests, a widely employed ensemble learning technique [11] for classification. Random Forests use multiple decision trees to enhance predictive accuracy and mitigate over-fitting. The parameters of the trees can be tuned, as well as the number of trees in the forest. To optimize the performance of the Random Forest Classifier, hyperparameter tuning was employed through the implementation of *RandomizedSearchCV*, a utility provided by the *scikit-learn* [10] library.

The parameter space is defined through the following distributions:

Parameter	Values
Max Features	[1, 2, 3, 5, None]
Max Leaf Nodes	[10, 100, 1000, None]
Min Samples Leaf	[1, 2, 5, 10, 20, 50, 100]
Min Samples Split	[2, 5, 10, 20, 50, 100]
Number of Estimators	[200, 300, 400, 800, 1000, 1400]
Max Depth	[1, 2, 3, 5, 10, 20, 50, 100, None]
Bootstrap	[True, False]
Criterion	["Gini", "Entropy"]

Table I: Hyperparameter selected search distributions.

The *RandomizedSearchCV* is used with specific configurations; 100 iterations, a 5-fold cross-validation strategy, and "F1 Macro" as the evaluation metric for scoring. The F1 score, particularly the macro variant, is chosen as it provides an accurate assessment of when the smallest class is under-performing.

After implementing the initial model, it was observed that it faced challenges in effectively handling class imbalances within the dataset, leading to suboptimal performance. Class imbalances often result in the model favoring the majority class and neglecting the minority class, impacting its ability to make accurate predictions for the underrepresented class. To address this issue, several methods of over and undersampling were employed to rebalance the class distribution in the training data.

One approach involved undersampling the majority class using the NearMiss technique from the imbalanced-learn library [12]. The NearMiss algorithm aims to reduce the number of instances from the majority class by selecting samples that are close to the decision boundary. By doing so, it helps mitigate the impact of class imbalance and promotes better generalization to the minority class. Random undersampling was also tried, with less than favorable results.

Additionally, oversampling techniques were utilized to augment the minority class instances. For this, BorderlineSMOTE, another method provided by imbalanced-learn, was used. BorderlineSMOTE focuses on generating synthetic samples for the minority class along the borderline between the minority and majority classes. This strategic oversampling should help in creating a more balanced dataset, enabling the model to better capture the patterns in the minority class.

None of the techniques described previously preformed better than simply dropping the rows with null values. Undersampling randomly or using Near Miss made the dataset too small. While oversampling decreased the predicting precision of the *Mitochondrial* class. The results discuss the most successful approach; using the hyperparameter tuning discussed earlier without any balancing. The subclasses were not attempted due to already poor performance on the *Genetic Disorder*.

B. Neural Network

The second model used is a *Keras TensorFlow* [13] Neural Network. The Sequential API provides simple MLPs that perform notably well. The model works as a linear stack of layers, where each layer has weights that are adjusted stochastically during training. The Multi-classification NN assignment formed the basis code for the implementation of this model and the tuning was executed with the help of *Hands On Machine Learning with Scikit-learn and TensorFlow* [14] book.

The network uses an input layer shaped to match the features of the training data, followed by four hidden layers with varying and decreasing numbers of neurons (400, 200, and 100). This setup was chosen after testing different increasing numbers of layers and neurons until the model stops improving or encounters issues like over-fitting. This happens when the training accuracy is remarkably larger than the validation and testing accuracies.

The activation functions used in these layers are Sigmoid and Relu. The Sigmoid function, denoted as $\sigma(x) = \frac{1}{1+e^x}$ is known for its characteristic S-shaped curve and is useful for binary classification, while Relu (Rectified Linear Unit), defined as $f(x) = max(0, x)$ helps to solve the vanishing gradient problem and speeds up training. Any combination of these two activations yielded the best results.

Two separate output layers are dedicated to each task: predicting the genetic disorder and its subclass. Both use a Softmax activation function, suitable for multi-class classification, which turns logits into probabilities that sum to one. The model applies L1 and L2 regularization to combat over-fitting, providing a balance between feature selection (L1) and shrinking coefficients (L2).

The model is compiled with the Adam optimizer, which is an ideal choice because the learning rate doesn't have to be set manually. It is initially set to default: 0.001. A good learning rate should be high enough to give precise slow convergence, but low enough to avoid overshooting minima or getting stuck in local minima. For this, an LR Reducing Callback was used during training for when the progress stalls.

The losses for both outputs are Categorical Cross Entropy, appropriate for multi-class classification problems. During training, several other callbacks are used: ModelCheckpoint to save the best model, EarlyStopping to prevent over-fitting,

and TensorBoard for visualization. The model is trained to a maximum of 100 epochs, with a batch size of 300. Thanks to EarlyStopping Callback, the model only runs for a few dozen epochs before it stops as it converges, the large batch size greatly helps with this fast convergence.

V. Results/Discussion

A. Random Forest

The Random Forest hyperparameter tuning resulted in the model parameters shown in Table II. This optimized model was evaluated using various performance metrics. The precision, recall and F1 score values were calculated for each class based on the predictions and true labels.

Hyperparameter	**Value**
Number of Estimators	800
Min Samples Split	20
Min Samples Leaf	10
Max Leaf Nodes	1000
Max Features	None
Max Depth	100
Criterion	'entropy'
Bootstrap	True

Table II: Optimized Random Forest Hyperparameters.

	Mitochondrial	Multifactorial	Single-gene	Average
Precision	0.74	0.40	0.59	0.58
Recall	0.75	0.38	0.59	0.57
F1 Score	0.74	0.39	0.59	0.58

Table III: Class-based and **macro** (average) metrics for Genetic Disorders.

To show the macro scores; row averages for precision, recall, and F1 score were computed in Table III. The averages are descriptive of the model's performance across all classes. The model demonstrated relatively high precision for the *Mitochondrial* category (0.74), indicating a strong ability to correctly identify instances of this type of disorder. However, precision was lower for *Multifactorial* (0.40) and *Single-gene* (0.59) disorders, suggesting some challenges in accurately classifying instances within these categories. The recall scores reveal a similar pattern, with higher values for *Mitochondrial* disorders (0.75) compared to *Multifactorial* (0.38) and *Single-gene* (0.59) disorders. The F1 scores, representing a balance between precision and recall, also follow this trend, with an overall average F1 score of 0.58. These results highlight the model's varied performance across different disease classes, especially with identifying the smallest class. This can be observed more clearly with a confusion matrix.

It is evident from Figure 4 that the model almost never predicts the *Multifactorial* class, and when it does, it cannot accurately distinguish it from the other two classes.

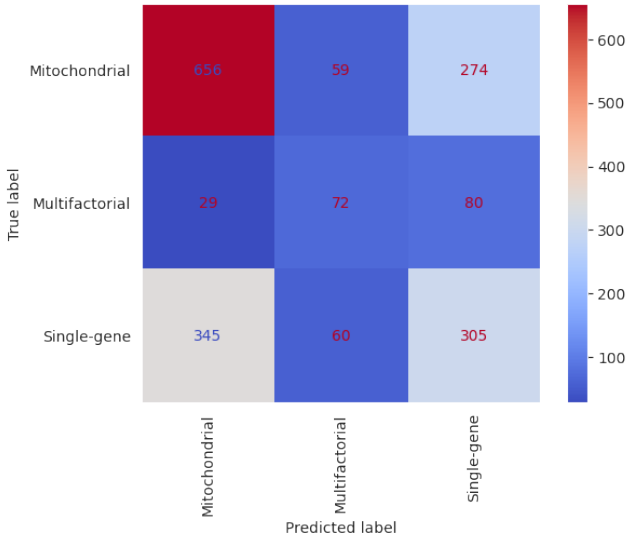


Figure 4: Confusion Matrix for Genetic Disorder (RF).

B. Neural Network

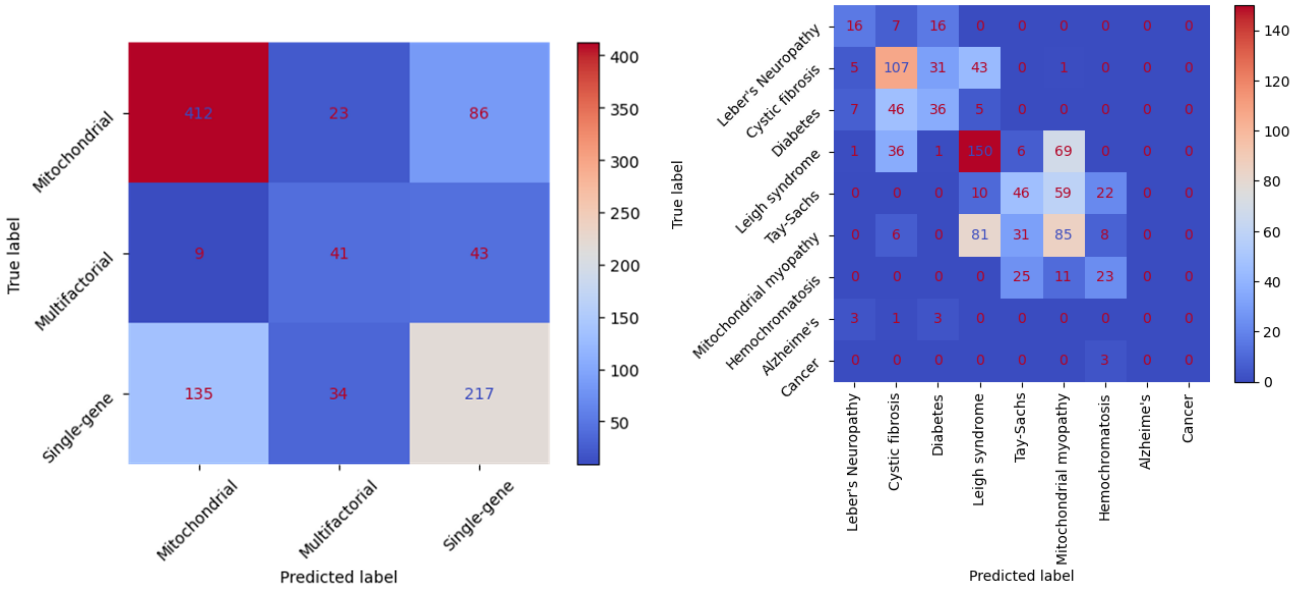
After extensive experimentation with prepossessing methods and adjusting the model's hyperparameters, the most effective settings are found, as described in section V. Table IV demonstrates that the model achieves a precision of 71% for *Genetic Disorder* and 56% for *Disorder Subclass*. These metrics are returned by Keras weighted, however, the macro metrics are more indicative of the model's performance. Table V presents the class-specific metrics for *Genetic Disorder*, focusing on accurately predicting the primary classes. Like the Random Forest model, the Neural Network tends to over-fit the predominant classes, namely Mitochondrial and Single-gene disorders. The macro averages for recall, precision, and F1 score are 60%, marginally surpassing the performance of the Random Forest model.

Metric	Genetic Disorder	Disorder Subclass
Precision	0.71	0.56
Recall	0.61	0.20
F1 Score	0.66	0.30
Accuracy	0.67	0.47
MSE	0.14	0.071
Loss	0.69	1.14

Table IV: **Weighted** performance metrics for *Genetic Disorder* and *Disorder Subclass* prediction.

	Mitochondrial	Multifactorial	Single-gene	Average
Precision	0.74	0.42	0.63	0.6
Recall	0.79	0.44	0.56	0.6
F1 Score	0.77	0.43	0.59	0.6

Table V: Class based and **macro** (average) metrics for the Genetic Disorders.



(a) Confusion Matrix for Genetic Disorder (MLP).

(b) Confusion Matrix for Disorder Subclass (MLP).

Figure 5: Confusion Matrices for both classes using an MLP.

The confusion matrix in figure 5 (a) highlights more the imbalance issue with the dataset, that results in poor performing metrics for the *Multifactorial* class. On the right (b), the matrix reveals that the model has a strong predictive accuracy for certain disorders, notably *Diabetes*. However, there are notable instances of misclassification, such as the confusion between *Tay-Sachs* and *Leigh syndrome*. This suggests that the model is struggling to distinguish between these subclasses, possibly due to similarities in the feature space or insufficient training data to differentiate them.

VI. Conclusion/Future Work

The dataset contains many features that are relevant to genetic disorders, yet they contain so many empty columns or outliers that it makes them almost useless to work with directly. The approach explained in the paper by Raza and collaborators is said to provide much better results. Nevertheless, little detail on the specific approach used to build the ETRF was given. The paper states that "The class predicted probabilities are extracted from the RF and ET techniques. A hybrid feature set is formed by combining the extracted class predicted probabilities. The hybrid feature set is later used as an input to applied learning techniques for predicting the genetic disorder and types of disorder." [1]. But they also note that "The total number of classifiers in the classifier chain is equal to the number of classes in the dataset used in this study" [1]. Therefore, it is not clear if the output from ETRF is the only thing used as input for the classifier chain or all the features are used.

Without the approach of the paper, some form of dimensionality reduction such as PCA, paired with better feature selection, could potentially yield better results for this study. A discriminant feature must be found to separate *Single-Gene* and *Multifactorial* diseases. Under and over-sampling was used to try and balance the dataset. While it did reduce over-fitting significantly, the overall performance of the classifier dropped.

Although the MLP performed slightly better than the RF model, it is evident from the confusion matrices that regardless of the model used, the issue is the same. They are over-fitting for the two more likely classes (*Mitochondrial* and *Single Gene*) and almost never predicting *Multifactorial*. This is probably because the selected features are very limited in their predictive power for that type of genetic disorder. Further analysis would require us to single out one feature that can distinguish *Multifactorial* from the rest.

VII. Contributions

Working on this project was immensely challenging, yet simultaneously enjoyable and intriguing. Going into the details of the models and trying to achieve our current solutions proved to be time-consuming. Initially, we began by experimenting with generic models using lazy-predict [15] before splitting onto separate paths to focus on the main models: Random Forest and Sequential Neural Networks. We tested various pre-processing methods for each model until we identified the most effective approaches. Throughout this process, we consistently discussed our progress, along with any notable observations regarding the dataset and its pre-processing.

Salma Loukili initially focused on the EDA, trying to understand the underlying data. She then proceeded to work on different Neural Networks until she found the most preforming one, MLP. The feature engineering she did was aimed at making that model preform as best as possible. She spent a significant amount of time writing the code to make the Keras models work, and then encoded the data in One-hot encoding to increase the performance of the MLP. When the performance was not satisfactory, she tried numerous combinations of feature selection, data cleaning, encoding, and MLP architecture. Finally, she made the graphs to visualize the performance of the MLP on both the *Genetic Disorder* and the *Disorder Subclass*.

Nicolas Gutierrez initially focused on finding the best features with the highest predictive power for the targets. A significant amount of time was spent trying to find features that were correlated in any way to the disorders. The hyperparameter tuning of the Random Forest and the over and under-fitting techniques was very interesting. Finally, he made the plots for the disorders and confusion matrix for the Random Forest.

References

[1] A. Raza, F. Rustam, H. U. R. Siddiqui, I. d. l. T. Diez, B. Garcia-Zapirain, E. Lee, and I. Ashraf, "Predicting genetic disorder and types of disorder using chain classifier approach," *Genes*, vol. 14, no. 1, 2023. [Online]. Available: https://www.mdpi.com/2073-4425/14/1/71

[2] "Of genomes and genetics : HackerEarth machine learning challenge," https://www.hackerearth.com/challenges/competitive/hackerearth-machine-learning-challenge-genetic-testing/, accessed: 2023-12-18.

[3] A. Raza, H. U. R. Siddiqui, K. Munir, M. Almutairi, F. Rustam, and I. Ashraf, "Ensemble learning-based feature engineering to analyze maternal health during pregnancy and health risk prediction," *PLOS ONE*, vol. 17, no. 11, pp. 1–29, 11 2022. [Online]. Available: https://doi.org/10.1371/journal.pone.0276525

[4] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Machine Learning*, vol. 85, no. 3, pp. 333–359, Dec 2011. [Online]. Available: https://doi.org/10.1007/s10994-011-5256-5

[5] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016. [Online]. Available: http://arxiv.org/abs/1603.02754

[6] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56 – 61.

[7] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[8] M. L. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021. [Online]. Available: https://doi.org/10.21105/joss.03021

[9] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[11] R. Polikar, "Ensemble learning," *Scholarpedia*, vol. 4, no. 1, p. 2776, 2009, revision #186077.

[12] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-365.html

[13] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[14] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Techniques and Tools to Build Learning Machines*. O'Reilly Media, Apr. 2017.

[15] S. R. Pandala, "Lazy predict," https://github.com/shankarpandala/lazypredict?tab=readme-ov-file, 2022.