UHASSELT
KNOWLEDGE IN ACTION

Maastricht University

Faculty of Sciences
School for Information Technology
Master of Statistics and Data Science

Master's thesis

Prediction of Pneumonia based on machine learning methods in Children Under Five in Uganda

ANAHITA HOSSEINKHANI

Thesis presented in fulfillment of the requirements for the degree of Master of Statistics and Data Science, specialization Data Science

SUPERVISOR :
Prof. dr. Olivier THAS

SUPERVISOR :
Dr. Dan KAJUNGU

2023
2024

Transnational University Limburg is a unique collaboration of two universities in two countries: the University of Hasselt and Maastricht University.

UHASSELT
KNOWLEDGE IN ACTION

www.uhasselt.be
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

Maastricht University

Faculty of Sciences
School for Information Technology
Master of Statistics and Data Science

Master's thesis

Prediction of Pneumonia based on machine learning methods in Children Under Five in Uganda

ANAHITA HOSSEINKHANI

Thesis presented in fulfillment of the requirements for the degree of Master of Statistics and Data Science, specialization Data Science

SUPERVISOR :
Prof. dr. Olivier THAS

SUPERVISOR :
Dr. Dan KAJUNGU

Master Thesis Data Science

Prediction of Pneumonia Based on Machine Learning Methods in Children Under Five in Uganda

Student:
Anahita Hosseinkhani (2262542)

Internal Supervisor :
Prof.Olivier THAS
External Supervisor :
Dan Kajungu

Contents

Abstract

Pneumonia remains the main cause of mortality among children under five years old in Uganda. This study aimed to identify the most effective machine learning model for predicting pneumonia in young children to enhance early diagnosis and treatment outcomes. A comprehensive comparison of six machine learning classification models, including Logistic Regression (LR), Support Vector Machine (SVM), Decision Tree (DT), K-Nearest Neighbor (KNN), Random Forest (RF), and Extreme Gradient Boosting (XGBoost), was conducted on an original and balanced dataset containing only symptoms and demographics of 7,859 infants, among whom 973 had pneumonia and 6,886 had non-pneumonia. Given that the dataset showed class imbalance, three methods of rebalancing were considered and compared: undersampling, oversampling, and SMOTE. Findings revealed that XGBoost (accuracy=95%, precision=86%, recall=73%, and F1 score=79%) and RF (accuracy=95%, precision=87%, recall=70%, and F1 score=78%) had the best performance on the original dataset. For the balanced datasets, considering undersampled, oversampled, and SMOTE methods, XGBoost, RF,and DT achieved the highest prediction results. Moreover, the predictors "respiratory distress" and "chest indrawing", combined with other predictors, were identified as the most significant features for predicting pneumonia across the original and all three balanced datasets. These results suggest that machine learning models can be used to accurately distinguish pneumonia in children in both balanced and imbalanced datasets. Additionally, the significance of key features remains consistent, highlighting their importance in predictive modeling for pneumonia detection.

1 Introduction

Pneumonia is an infection of the lungs that remains one of the major causes of mortality especially among children under five years of age worldwide[1]. Despite advances in medical science, the timely and accurate diagnosis of pneumonia in young children remains a critical challenge, particularly in low-resource settings. Early diagnosis and treatment are essential to improve outcomes and reduce the burden on healthcare systems. Effective diagnosis is crucial because it allows for timely intervention with appropriate antibiotics and supportive care, significantly reducing the risk of complications and death. However, traditional diagnostic methods often rely on clinical signs and symptoms, such as fever, cough, and difficulty breathing, which can be nonspecific and easily confused with other respiratory infections. Furthermore, radiographic imaging and laboratory tests, while more definitive, often have low sensitivity for detecting early-stage pneumonia and can cause potential harm due to X-ray exposure. Additionally, the absence of abnormalities in chest radiographs in some children further complicates their diagnostic utility[2]. These factors, coupled with the lack of availability in low-resource settings, frequently result in delays in diagnosis and treatment. Consequently, there is a crucial need for innovative approaches to enhance the predictive accuracy of pneumonia diagnosis.

In recent years, the integration of machine learning and data science techniques in healthcare has shown significant potential in improving diagnostic processes and patient outcomes[3]. Machine learning models can analyze large amounts of clinical data to uncover patterns and associations that may not be immediately apparent to human clinicians[4]. These models can help in making more accurate predictions, thereby supporting healthcare professionals in their decision processes. Moreover, Machine learning has proven effective in enhancing diagnostic accuracy for pneumonia, particularly for adult patients[5].

This thesis aims to identify the most effective machine learning model for predicting pneumonia in children under five years old in Uganda. To achieve this objective, a comprehensive comparison of

six machine learning classification models, including Logistic Regression (LR), Support Vector Machine (SVM), Decision Tree (DT), K-Nearest Neighbor (KNN), Random Forest (RF), and Extreme Gradient Boosting (XGBoost), were utilized based on the symptoms and demographics features. Each of these models represents a distinct approach to classification tasks and has been extensively employed in various domains for their robustness and predictive capabilities[6].

The dataset for this study includes a different range of features that are hypothesized to be relevant for pneumonia prediction. These features include demographic details, clinical symptoms, and condition factors. The data has been cleaned and preprocessed to ensure its quality and reliability, forming a robust foundation for developing and evaluating the machine learning models.

By comparing the performance of these models using metrics such as accuracy, precision, recall, and the f1 score, this research aims to identify the best model that offers the highest predictive performance based on the symptoms and demographics features. Finally, the chosen model will help healthcare professionals in making more accurate and timely diagnoses, potentially leading to better patient outcomes and more efficient use of medical resources.

2 Description of the dataset

The dataset used in this study was retrospectively collected to cover a period of six years study from 2014 to 2019 at a district hospital in Uganda. This data was provided in Excel format, to access easily. It included a wide range of features that were hypothesized to be relevant for pneumonia prediction. The full dataset consisted of 101 features. These features included demographic details (age, gender, village name), clinical symptoms (respiratory distress, chest in-drawing, presence of malaria, skin pustules, body temperature, ability to drink water, cyanosis), and conditions of infants (used zinc supplements, amoxicillin). Each of these variables was recorded and validated to ensure data quality and reliability. Additionally, there were two variables indicating the pneumonia status of children aged 5 years and below: "severe pneumonia" and "pneumonia". Among all variables, only those related to village name and gender were categorical, and age was continuous while the rest of the features were binary.

The dataset initially included 7,880 infants. Since the primary objective of this study was to predict pneumonia in children under five, data exploration identified 21 infants outside this age range, which were considered outliers and removed from the dataset. Handling missing data was a critical aspect of this study. Initial exploration revealed some missing values in certain features, such as village name and patients id. Specifically, 135 records for the patients id were labeled as "not recorded". To avoid losing information, this variable was excluded from the analysis. The village name variable had 23 records with "not recorded" values, since this variable was categorical, these records were reassigned to "None".

Given that machine learning models require numerical variables, the data underwent preprocessing, including encoding categorical variables such as gender and village name. Moreover, the two variables "severe pneumonia" and "pneumonia" were aggregated into a single variable named "Pneumonia."

After completing data cleaning and preprocessing, the final dataset used in this study included 7,859 infants, consisting of 3,989 boys and 3,870 girls, with ages ranging from 0.03 months (1 day) to 60 months. Among these infants, 12% of the instances (973 cases) were identified as pneumonia cases, while the remaining 88% (6,886 cases) were distinguished as non-pneumonia cases.

3 Methodology

In this study, the primary objective is to identify the most effective machine learning model for predicting pneumonia. To achieve this goal first, a comprehensive comparison of several classification models like LR, SVM, DT, KNN, RF, and XGBoost, was conducted. Each of these models represents a distinct approach to classification tasks and has been extensively employed in various domains for their robustness. Given that the primary goal is binary classification, these algorithms were specifically chosen for their strengths in this area.

LR is one of main techniques in statistical modeling which can be used for predicting binary outcomes. It is designed to handle dichotomous outcomes by modeling the probability of a particular class or event. So,in order to model the binary probability, the logistic function is utilized, which is defined as:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

The quantity $\frac{p(X)}{1-p(X)}$ is called the odds and can take on any value between 0 and ∞. Values of the odds close to 0 indicate very low probabilities, while values close to ∞ indicate very high probabilities. By taking logarithm of both side LR model can be employed. Since a linear relationship makes it easier to interpret the impact of each feature on the response variable, the LR can be interpreted in terms of log odds, the left-hand side of the equation is called log odds or logit. So, LR has the form of linear in terms of logit[6].

$$\log\left(\frac{P(Y = 1 \mid X)}{1 - P(Y = 1 \mid X)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$$

LR provides a simple and interpretable framework for classification,it is also computationally efficient, and works well with relatively small datasets.

SVMs are known for their ability to create robust binary classification boundaries and handle high-dimensional data efficiently. They are effective when the number of dimensions exceeds the number of samples, and also robust to overfitting by maximizing the margin, especially in high-dimensional space. They are a powerful extension of the support vector classifier that utilizes kernels to enhance the feature space, which helps in adjusting nonlinear boundaries, between different classes.

SVM constructs a hyperplane or set of hyperplanes in a high-dimensional space to separate data points that can not be separate linearly into different classes. A data point is represented as a p-dimensional vector, and the objective is to find a $(p-1)$-dimensional hyperplane that separates the classes. Particularly, when the data is not linearly separable, the feature space can be expanded using kernel functions.

When the boundary between two classes is linear , a support vector classifier(SVC) approach is used. A linear boundary, or decision boundary, is a straight line that separates the two classes in the feature space. The SVC aims to find the optimal linear boundary that maximizes the margin between the classes, which is the distance between the closest points of each class (called support vectors) and the boundary itself. This maximized margin helps to improve the classifier's ability to generalize to new data. It relies on the concept of inner products. The inner product of two vectors **a** and **b** is defined as:

$$\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{r} a_i b_i$$

For two observations $\mathbf{x}_i$ and $\mathbf{x}_{i'}$, the inner product is given by:

$$\langle \mathbf{x}_i, \mathbf{x}_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

The linear support vector classifier can be expressed as:

$$f(\mathbf{x}) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle$$

To evaluate function f(x) the inner product of new point x and each training points $\mathbf{x}_i$ is needed. In order to handle non-linear boundaries, SVM approach is used. So, the inner product is replaced with a kernel function K. A kernel is a function that quantifies the similarity between two observations. By using different kernel functions, more flexible decision boundaries are created. A popular choice is the polynomial kernel:

$$K(\mathbf{x}_i, \mathbf{x}_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij} x_{i'j} \right)^d$$

where d is the degree of the polynomial. This kernel enables the SVM to fit a polynomial decision boundary in the original feature space, effectively transforming it into a higher-dimensional space.

Another widely used kernel is the radial basis function (RBF):

$$K(\mathbf{x}_i, \mathbf{x}_{i'}) = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \right)$$

where γ is a positive constant. The RBF kernel has the advantage of creating a local influence, meaning that only nearby training observations affect the classification of a new point. The use of kernels allows SVMs to operate in an implicitly defined high-dimensional space without explicitly computing the coordinates in that space. This is particularly beneficial for kernels like the RBF, where the feature space can be infinite-dimensional, making direct computation impractical.

DTs are among the most intuitive and interpretable models in machine learning. They are able to decompose complex decision-making processes into a series of simpler decisions, creating a

tree structure that is easy to understand and visualize[6]. Each tree includes a root node, which serves as the topmost node and the starting point for splitting the data based on feature values. Internal nodes are intermediate nodes in the tree where the predictor space is split based on certain conditions. Branches that connect these nodes. Lastly, leaf nodes, also known as terminal nodes, represent the final decision or outcome. In order to make a classification tree, each observation from the dataset is assigned to a specific region, based on the decision rules learned during the training process. Once an observation reaches a terminal node, the classification tree predicts that the observation belongs to the most commonly occurring class within that region.
The task of growing the classification tree is based on three methods: classification error rate which is the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk})$$

The symbol pˆmk indicates the proportion of training observations in the m-th region that belong to the k-th class. However, classification error is not sensitive enough for growing decision trees effectively. In practice, two other measures, the Gini index and entropy, are preferred.

The Gini index is defned by:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

It measures total variance across the K classes. If all of the pˆmk's are close to zero or one then the Gini index takes on a small value. An alternative to the Gini index is entropy, which given by:

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

Given that $0 \leq \hat{p}\_mk \leq 1$, it follows that, $0 \leq -\hat{p}\_mk \log \hat{p}\_mk$.

Entropy will take on a value near zero if the pˆmk's are all near zero or near one. So, to build a classification tree, either the Gini index or entropy can be used to evaluate the quality of a specific split. While trees are easy to explain and display graphically,but they are very nonrobust and small changes cause a large changes in the final estimated trees. However, by using methods like RF and XGBoost the predicted performance of trees can be improved.

Ensemble methods like RF and XGBoost are a group of techniques that combine multiple simpler models to create a single powerful model. RF is an improvement over a method called bagging. Bagging involves building several decision trees on bootstrapped samples of the training data. However, in RF, for constructing DTs, each time a split is considered, only a random subset of features is chosen. The algorithm is not allowed to consider a majority of the available predictor and the number of predictors considered at each split is approximately equal to the square root of the total number of predictors, $m \approx \sqrt{p}$ which m is a number of predictors at each split and p indicates a total number of predictors.

In a regular bagged tree, most or all of the trees might end up using the strong predictor for the first split. This makes the trees very similar to each other,and predictions from the bagged trees are

highly correlated. Consequently,averaging over all trees will not help in reducing the variance. In particular, this means that bagging will not lead to a substantial reduction in variance over a single tree in this setting. So, RF solves this problem by limiting the number of predictors considered at each split. This means that, on average, only a fraction of the predictors will be used for each split, and (p-m)/p of the splits will not consider the strong predictors, giving other predictors a chance to contribute to the decision-making process. Therefore, RF decorrelates the trees, making them less similar to each other and reducing the overall variability in the model's predictions. This helps to prevent overfitting and makes the model more reliable.

XGBoost stands out as one of the most powerful and popular algorithms in machine learning. It follows the gradient boosting framework, which involves sequentially adding DTs to the model, each tree is fit on a modified version of the original dataset. In this method, the tree is fitted using the residuals rather than the original outcome. In other words, in each iteration, a DT is fitted using the current residuals as the response variable instead of the original outcome variable.

As mentioned before, one of the advantages of trees is their easily interpreted visual representation. However, when employing the technique of bagging, which involves aggregating a large number of DTs, it becomes impractical to represent the resulting model with a single tree. Therefore, while bagging significantly enhances prediction accuracy, it does at the cost of model interpretability. Although, it is hard to explain bagged trees but overall summary of important features of each predictor can be obtained using the Gini index. So, in order to determine the importance of each predictor in bagging classification trees, the total reduction in the Gini index is achieved by splits over a given predictor, and this value is then averaged over all trees in the ensemble.

KNN is one of simplest machine learning algorithm. It first identidifies k points close to x_0, test observations. For each test observation x_0, the k-NN classifier calculates the distance between x_0 and all points in the training dataset. The K points with the smallest distances are selected as the nearest neighbors, represented by N_0. The classifier then estimates the conditional probability for each class j. This is done by calculating the fraction of the K nearest neighbors which response values equal j. Mathematically, this is represented as:

$$\Pr(Y = j \mid X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j)$$

One effective method to increase the performance of all machine learning models is through hyperparameter tuning. Hyperparameter tuning, also known as hyperparameter optimization, is the process of finding the best set of hyperparameters for a machine learning model. It is essential because the performance of machine learning models can significantly depends on the choice of hyperparameters and this hyperparameters need to be fixed before training. In this study, grid search has been utilized to find hyperparameters for machine learning models. This method involves exhaustively searching through a manually specified subset of the hyperparameter space [6]. Defining a grid of hyperparameter values and for each combination, the model is trained and evaluated using stratified 5_fold cross-validation. In Table 4 summary of each hyperparameters used in this study for different models reported.

So, by comparing the performance of these six machine learning models in predicting pneumonia, the model shows the highest accuracy, precision, recall, and F1 score will be determined. The findings gained from this comparative analysis will help in selecting the best model for the

prediction of pneumonia. By analyzing these evaluation metrics, a comprehensive understanding of each model's strengths and weaknesses is gained. For example, a model with high precision but low recall may be excellent at identifying pneumonia but may miss many actual cases. In contrast, a model with high recall but low precision may identify nearly all pneumonia cases but also incorrectly classify many non-pneumonia cases as pneumonia.

3.1 Evaluation Metrics

In binary classification tasks, it is essential to use multiple criteria for evaluating the performance of predictive models. Therefore, this study compared the performance of six models using a comprehensive set of metrics, including F1 score, accuracy, recall, and precision.

These performance metrics were calculated based on the counts of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). For instance, accuracy represented the proportion of correctly predicted samples to the total samples, providing a measure of overall model performance. The recall measured the proportion of positive samples correctly identified as positive out of the total number of positive samples. It indicates how many of the positive class samples present in the dataset were correctly identified by the model. Given the importance of accurately identifying positive instances in this study, recall served as a critical metric for evaluating model effectiveness. precision quantifies the proportion of correctly identified positive instances among all instances that the model classified as positive. A high precision rate indicated a low rate of false positives, which is desirable in predictive modeling.

In binary classification, a perfect predictive model would ideally indicate both high recall and precision rates. However, therefore often exists a trade-off between these metrics, as well as with accuracy. For example, increasing precision may lead to a decrease in recall and vice versa. This trade-off becomes particularly crucial in scenarios where both precision and recall are equally important, and neither can be favored at the expense of the other. In such cases, the F1 Score stands out as the primary metric of choice. The F1 Score provides a comprehensive evaluation by computing the harmonic mean of recall and precision, offering a balanced assessment of model performance. Unlike simple averages, the harmonic mean penalizes extreme values, ensuring that a high F1 Score is only achieved when both precision and recall are high[7].

In the context of pneumonia prediction, high recall ensures that most cases of pneumonia are correctly identified, minimizing the risk of false negative cases. High precision, on the other hand, ensures that the cases identified as pneumonia are indeed true positives, reducing the burden of unnecessary treatments and follow-up tests on patients who do not have the disease. Thus, the F1 Score is particularly valuable in medical diagnostics, where the consequences of false negatives (missed diagnoses) and false positives (incorrect diagnoses) can both be significant. By using the F1 Score as the primary metric, the evaluation process captures the model's ability to balance precision and recall, leading to more reliable and actionable predictive outcomes. For clarity, Table 1 presents the equations utilized for calculating each of the performance metrics: accuracy, precision, recall, and the F1 score. These metrics provided a robust framework for evaluating the efficacy and reliability of the predictive models employed in this study.

Additionally, a confusion matrix was utilized, which visually represents the model's predictions

each dataset, and also helps to have better understanding of evaluation metrics. In binary classification, it displays counts of predicted and actual values, helping to evaluate the model's performance. Figure 1 displays"TN" as accurately classified negative examples, while "TP" indicates accurately classified positive instances. Conversely, "FP" represents actual negatives classified as positives, and "FN" denots actual positives classified as negatives.

There are several methods for evaluating performance metrics in machine learning, the most common are train/test splits and cross-validation. The simplest model evaluation procedure is to split a dataset into two parts and use one part for training a model and the second part for testing the model. These parts of the dataset are named according to their function, train set, and test set, respectively. This method is effective if collected dataset is very large and representative of the problem. It is easy to implement and understand. Also, it is less computationally intensive compared to other methods like cross-validation, making it faster, especially for large datasets. One of the reasons that train test split was not used in this study is that considering the common ratio of train test split like 70% for training and 30% for test, then this split might result in a test set that does not accurately reflect the class distribution of the overall dataset. In other words, the split may have few examples of the minority class, if the minority class is underrepresented or even missing from the test set, the performance metrics will not reflect the model's ability to handle the minority class. So, this unbalanced distribution in the test set can lead to misleading performance metrics, where the model might appear to perform well overall but actually fails to predict the minority class correctly. Therefore, with an imbalanced dataset, a model could achieve high accuracy by simply predicting the majority class for all instances. If the test set reflects this imbalance, then accuracy as a metric becomes less informative and overly optimistic. Also, other metrics like precision, recall, and F1 score for the minority class can be significantly skewed, giving a false sense of the model's effectiveness.

The other common method for evaluating performance metrics of machine learning models is k-fold cross-validation. This procedure involves randomly splitting the set of observations into k folds. The first k-1 folds are used to train a model, and the holdout kth fold is used as the test set[6]. This process is repeated and each of the folds is given an opportunity to be used as the holdout test set. A total of k models are fit and evaluated, and the performance of the model is calculated as the mean of these fits. By averaging the performance over these multiple splits, the method reduces the impact of any one particularly good or bad split. This leads to a more reliable and less optimistic estimate of model performance since every data point gets to be in the test set once and in the training set k-1 times, ensuring that the performance evaluation is based on all available data. Although this might work fine for data with a balanced class distribution, when the distribution is severely skewed, one or more folds will likely have few or no examples from the minority class. So, in that situation, stratified cross-validation is preferred. In this study, since the original dataset was imbalanced, 5-fold stratified cross-validation was used which is a variation of cross-validation. This variation is designed to handle imbalanced dataset or those with a skewed class distribution. In the stratified k-fold cross-validation, each fold is made by preserving the same percentage of samples for each class as in the complete dataset. This ensures that each fold has a similar class distribution, leading to more accurate performance estimates. This approach makes sure that each class is adequately represented across all folds, allowing for a fair and accurate evaluation of the model's performance.

In balanced datasets, where each class is equally represented, 5-fold stratified cross-validation was employed to ensure fair comparisons with the results of the original dataset. Although the benefits of stratification are less pronounced but still useful. However, stratified k-fold cross-validation generally does not cause optimistic results in balanced datasets. Instead, it ensures that the evaluation is fair and reliable by maintaining consistency in class distribution across folds. Ensures that each fold has a consistent representation of classes, leading to reliable performance metrics. This prevents the random bias that might occur if some folds have a different class distribution, which could either inflate or deflate performance metrics.

Table 1: Evaluation Metrics

Metric	**Equation**
Accuracy	$\frac{TP+TN}{TP+FP+FN+TN}$
Recall	$\frac{TP}{TP+FN}$
Precision	$\frac{TP}{TP+FP}$
F1 Score	$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision}+\text{Recall}}$

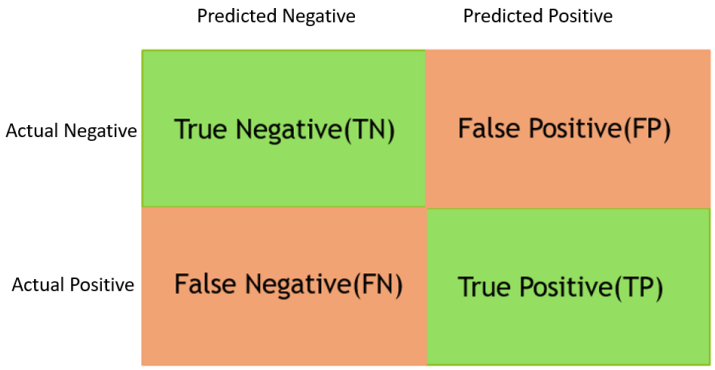

Figure 1: Confusion matrix

3.2 Imbalance Data Set

The dataset used in this study showed a notable degree of class imbalance which is a common occurrence in classification tasks. Imbalance refers to the situation where the distribution of classes within the dataset is highly skewed, with some classes representing a significantly larger proportion of the data than others. Specifically, classes that include a large proportion of the dataset are termed 'majority classes,' while those representing a smaller fraction are referred to as 'minority classes.

The severity in the minority class can vary, ranging from mild, where the minority class comprises 20–40% of the dataset, to moderate, where it represents 1–20%, and extreme, where it constitutes less than 1% of the dataset[7]. In the dataset analyzed for this study, the minority class is considered

moderate, including only 12% of instances. By rebalancing class distributions using the sampling method, machine learning models are able to learn from a balanced dataset, therefore improving their performance in predicting minority class instances.

Addressing class imbalance is essential in order to prevent bias in machine learning models[8]. Failure to adequately account for this issue can reflect on model performance lead to poor predictions. Therefore, it is essential to implement strategies to rebalance the dataset prior to applying machine learning algorithms[9].

Resampling techniques help as a fundamental strategy for addressing the challenges caused by highly unbalanced datasets. In this study three techniques have been utilized: simple random under-sampling, over-sampling, and Synthetic Minority Over-sampling Technique (SMOTE). Simple random undersampling is one of the common techniques for resampling imbalanced data[10]. It reduces the dominance of the majority class by removing instances from the majority class. In contrast, over-sampling which requires adding more examples from the minority class, aims to strengthen the representation of the minority class, ensuring that it receives adequate attention during model training. SMOTE, addresses the limitations of traditional over-sampling methods by generating synthetic samples rather than simply duplicating existing ones. This technique creates new minority class instances by interpolating between existing ones, thereby preserving the underlying distribution of the minority class while reducing the risk of overfitting[8]. In this methodology, the process begins by selecting a random example from the minority class. Subsequently, the algorithm identifies the k nearest neighbors for this selected example, often setting k to a value such as 5. From this set of neighbors, one is randomly chosen. Then, a synthetic example is generated at a point randomly selected between the chosen example and its selected neighbor within the feature space. This approach effectively expands the dataset by creating synthetic instances that closely resemble the minority class which addresses the imbalance issue and enhances the model's ability to learn from underrepresented samples.

Figure 2: Random undersampling method Figure from [8]

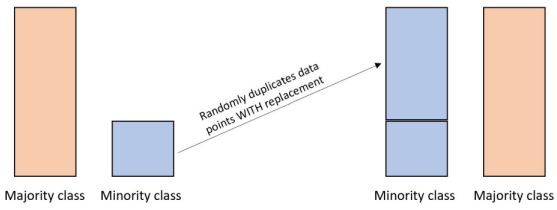

Figure 3: Random oversampling method Figure from [8]

However, it is essential to recognize that while resampling techniques offer benefits in terms of balancing classes, they are not without limitations. One main drawback of under-sampling strategies is randomly removing records from the majority class, which can result in the loss of valuable information. This random removal may cause removing important patterns and information of the majority class. Moreover, the sample chosen may not be an accurate representation of the population, which leads to a biased sample. Additionally,over-sampling increases the risk of overfitting, where the model becomes overly specialized to the training data, leading to poor generalization on unseen data. This occurs when random records from the minority class are duplicated, potentially raising the importance of certain instances and compromising the model's ability to generalize to new data. Although, SMOTE reduces the risk of overfitting, but it doesn't consider the majority class while creating synthetic samples. In addition to that, it may create synthetic samples between samples that represent noise. As a result, the new dataset will have more noise than the original one, which can affect performance.

So, in this study, the machine learning models were applied in two distinct phases to increase the robustness and accuracy of pneumonia prediction. First, all machine learning models were tested on the original dataset, which included only demographic details and clinical symptoms. Second, all machine learning models were then evaluated using three different rebalancing techniques, undersampling, oversampling, and SMOTE to address class imbalances and improve model performance. Each of these phases helped to determine the most effective machine learning model and preprocessing strategy for predicting pneumonia in young children.

4 Results and Discussion

The study comprising of total 7,859 infants, including 3,989 boys and 3,870 girls, with ages ranging from minimum of 1 day to 60 months. As shown in Figure 4, the number of pneumonia cases decreases as infants get older. Preliminary data exploration revealed that approximately 12% of the entire dataset exhibited pneumonia cases (973 instances) and around 88% percent of non-pneumonia cases (6886 instances). The distribution of pneumonia cases can be seen in Figure5.

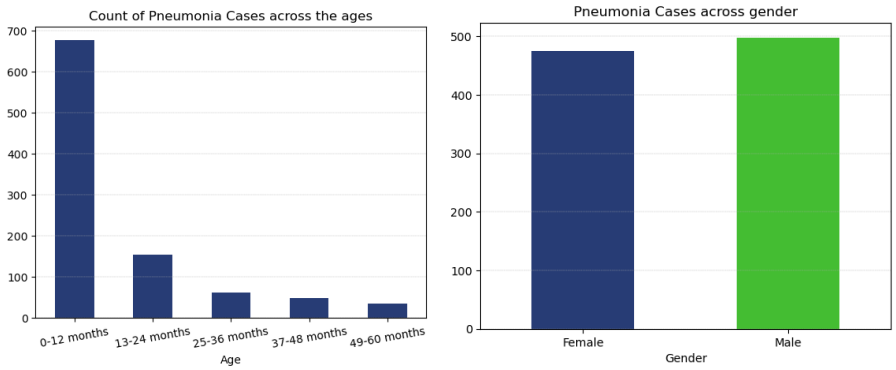

Figure 4: Number of pneumonia cases across different age and gender

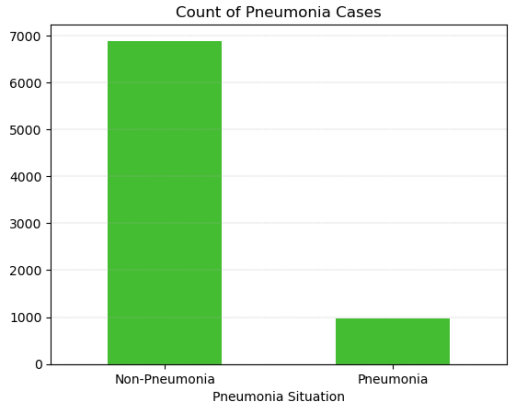

Figure 5: Distribution of pneumonia cases

Moreover, to address the class imbalance in the dataset, resampling techniques were employed. This process yielded four distinct datasets: the original imbalanced dataset, the undersampled dataset, the oversampled dataset, and the SMOTE dataset. These datasets served as the foundation for subsequent analyses and model development. Furthermore, a confusion matrix, which visually represents the model's predictions for each dataset, was used.

4.1 Evaluation of performance with the original dataset

In this part, the findings from the comparison of six machine learning models on the original dataset, considering only demographics and symptoms, are reported in Table 2. The results for RF and XGBoost indicated close performance.

The XGBoost model demonstrated the highest performance (accuracy = 95%, precision = 86%, recall = 73%, and F1 score = 79%), followed by the RF model (accuracy = 95%, precision = 87%, recall = 70%, and F1 score = 78%) and the DT model (accuracy = 94%, precision = 84%, recall = 72%, and F1 score = 76%). The F1 score for the XGBoost model reached 79%, while for RF and DT, it stood at 78% and 77%, respectively. Figure 6 shows the confusion matrix for both RF and XGBoost on the original dataset. It is evident that the RF model wrongly predicted more cases (292 cases) compared to the XGBoost model (264 cases).

In order to understand the importance of each predictor, Gini importance or mean decrease in impurity (MDI) was calculated. It counted the number of times a feature was used to split a node, weighted by the number of samples it split [6]. Figure 7 displays the important features for both the RF and XGBoost models. It demonstrates that among all the trees in the RF and XGBoost models, the predictors "respiratory distress" and "chest indrawing with fast breathing" were the most influential variables.

Table 2: Performance comparison of machine learning models on the original dataset

Model	Recall	F1 Score	Accuracy	Precision
RF	70%	78%	95%	87%
XGBoost	73%	79%	95%	86%
DT	72%	77%	94%	84%
LR	72%	76%	86%	95%
SVM	68%	75%	84%	94%
KNN	65%	72%	82%	92%

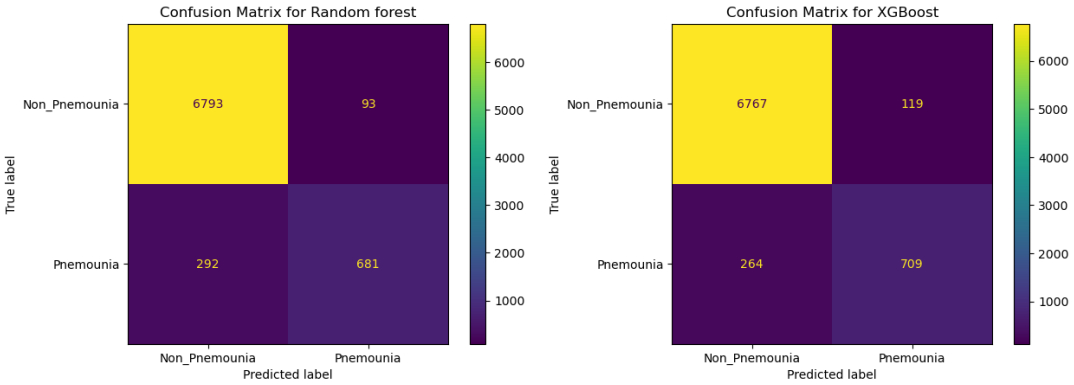

Figure 6: Confusion matrix for RF and XGBoost on the original dataset

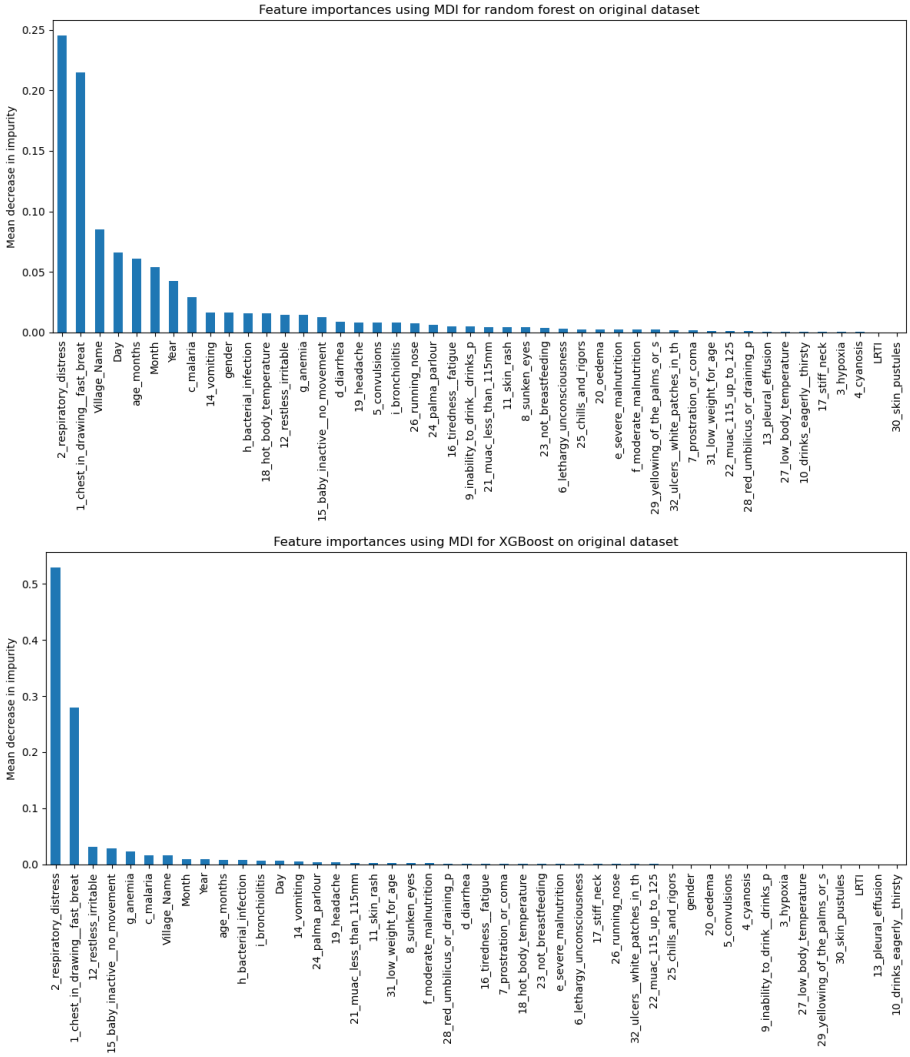

Figure 7: Feature importance for RF and XGBoost on the original dataset

4.2 Evaluation of performance with the balanced datasets

Given the fact that the dataset indicates a huge class imbalance, with 973 instances of pneumonia and 6,886 instances of nonpneumonia, it was required to address this bias. To achieve this, three different methods undersampling, oversampling, and SMOTE were utilized to balance the dataset. The results of these approaches were then compared to determine their effectiveness.

In undersampling, the number of instances for each class was 973, achieved by randomly removing the instances in the majority class to match the minority class. Table 3 presents a comparison of six machine learning models on an undersampled dataset. The results highlight that RF and XGBoost had the highest performance among the models evaluated. Specifically, the F1 score for

RF and XGBoost both achieved scores of 87%.

Figure 8 visualizes the confusion matrices of the RF and XGBoost models for the undersampled dataset. It is evident that the number of cases incorrectly identified as nonpneumonia is 190 in the XGBoost model, compared to 170 cases in the RF model. Although both models showed close results this small discrepancy can be attributed to the inherent differences in how these models handle classification tasks. The RF model, which is an ensemble of decision trees, tends to provide more robust predictions by averaging the results of multiple trees. This averaging process can reduce the variance and help the model generalize better on the undersampled dataset, leading to slightly fewer false negatives.

The MDI displays which features were important for RF and XGBoost models on the undersampled dataset. Figure 9 highlights that the predictors "respiratory distress" and "chest indrawing with fast breat" were critically important for both models. Following these, "malaria" and "village name" were notably important for the XGBoost model, while "village name" and "age of the child" were key predictors for the RF model. This distinction underscores how each model evaluates the contribution of different features to its predictive performance. For XGBoost, the inclusion of "malaria" as a significant predictor suggests its utility in capturing complex interactions between co-infections and pneumonia. In contrast, RF places more emphasis on the "age of the child", indicating its broader consideration of demographic factors. Both models agree on the high importance of "respiratory distress" and "chest in drawing fast breat", reflecting these symptoms have critical role in diagnosing pneumonia.

In the oversampling method, the number of instances for each class was adjusted to 6,886 by resampling from the majority class. Table 3 displays a comparative analysis of six machine learning models using an oversampled dataset. The findings underscore the superior performance of RF and XGBoost among the rest of the models,achieving the highest F1 score at 98%, followed by DT which reach scores of 97%. However, the slight difference in the overall F1 scores (98% for RF and 97% for DT) highlights the added benefit of aggregation of multiple DT in RF and XGBoost which leads to better generalization and slightly improved performance metrics, such as precision and recall, which contribute to a higher F1 score. Figure10 shows the confusion matrices of the RF and XGBoost models on the oversampled dataset. The number of cases incorrectly identified as non-pneumonia for RF and XGBoost models is equal to 5 and 12 cases,respectively.

Table 3: Performance comparison of machine learning models on different datasets

	Recall	F1 Score	Accuracy	Precision
Undersampled Method				
RF	83%	87%	88%	92%
XGBoost	80%	87%	88%	94%
DT	81%	85%	86%	91%
LR	81%	86%	86%	92%
SVM	82%	86%	87%	93%
KNN	63%	63%	64%	64%
Oversampled Method				
RF	99%	98%	97%	98%
XGBoost	99%	98%	98%	97%
DT	99%	97%	96%	94%
LR	81%	86%	87%	92%
SVM	81%	87%	87%	93%
KNN	98%	94%	93%	88%
SMOTE Method				
RF	94%	94%	94%	93%
XGBoost	96%	95%	95%	95%
DT	93%	91%	91%	90%
LR	88%	89%	89%	89%
SVM	84%	87%	87%	90%
KNN	95%	88%	87%	82%

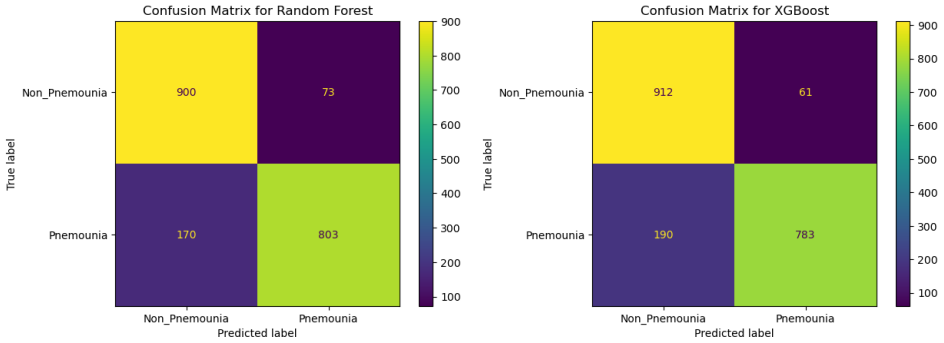

Figure 8: Confusion matrix for RF and XGBoost on the undersampled dataset.

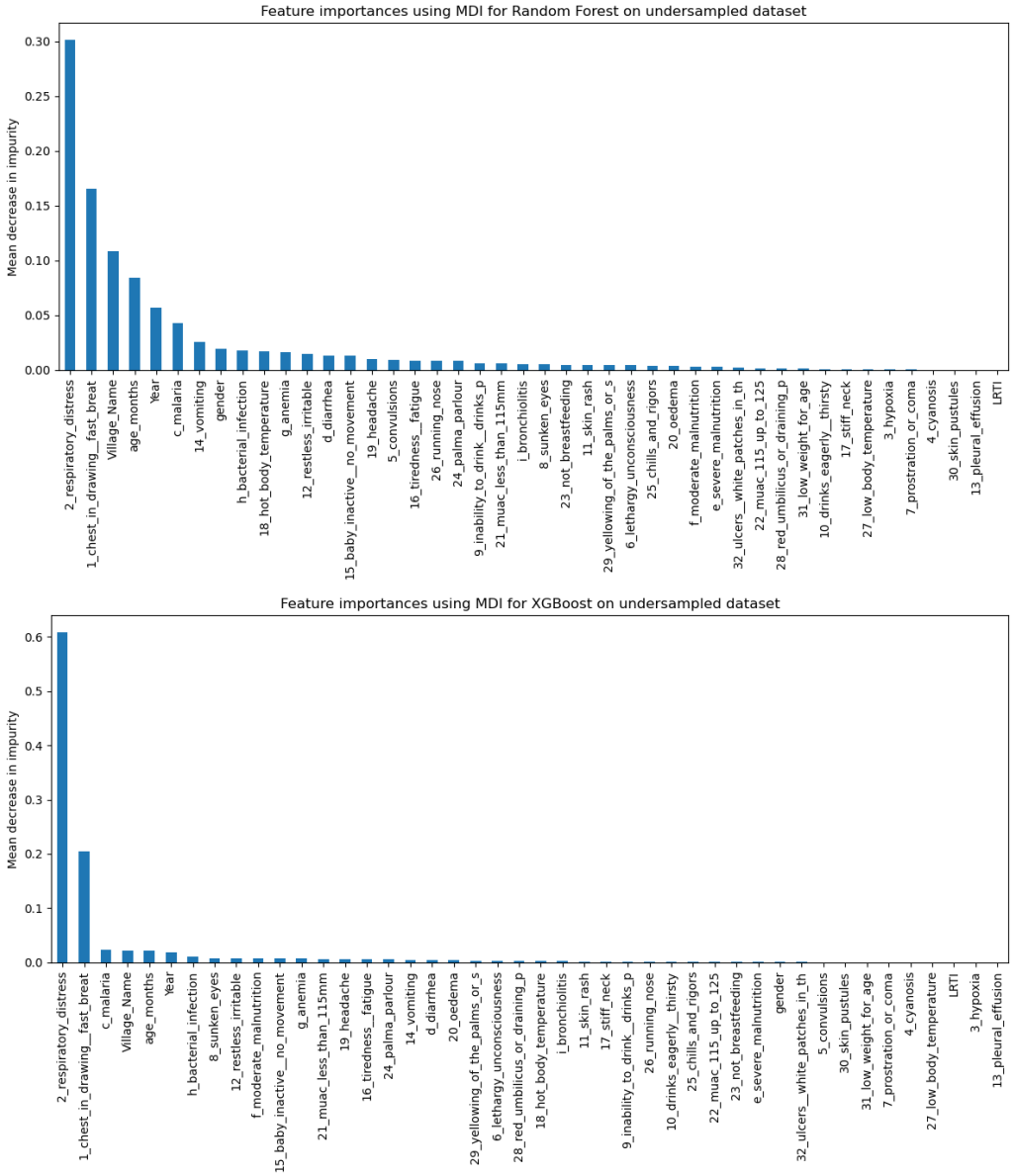

Figure 9: Feature importance of RF and XGBoost on the undersampled dataset.

Figure 11 displays the important features for both the RF and XGBoost models on the oversampled dataset. It highlights that the predictors "respiratory distress" and "chest indrawing with fast breathing" again are crucial for both models. However, there are some differences in the important features identified by each model. For the RF model, the predictors "village name," "age," and "malaria" were considered significant. These features played a critical role in the model's decision-making process. The importance of these features suggests that the RF model placed significant

emphasis on demographic and symptom features in its prediction process.

In contrast, the XGBoost model identified different important features. The predictors "low weight", "muac_less_than_115mm", "red umbilicus", and "low temperature" were distinguished as essential for the XGBoost model. These features had a high influence on the model's performance and were prioritized in its predictions. "muac (mid-upper arm circumference) less than 115mm" and "low weight" are indicators of severe malnutrition, while "red umbilicus" and "low temperature" are clinical signs that indicate infection. The XGBoost model's focus on these predictors suggested it is more sensitive to symptoms indicators. So, while both models agreed on the importance of certain predictors related to respiratory symptoms, they differed in their emphasis on other features. The RF model focused more on demographic and symptom features, whereas the XGBoost model gave more weight to symptom indicators.

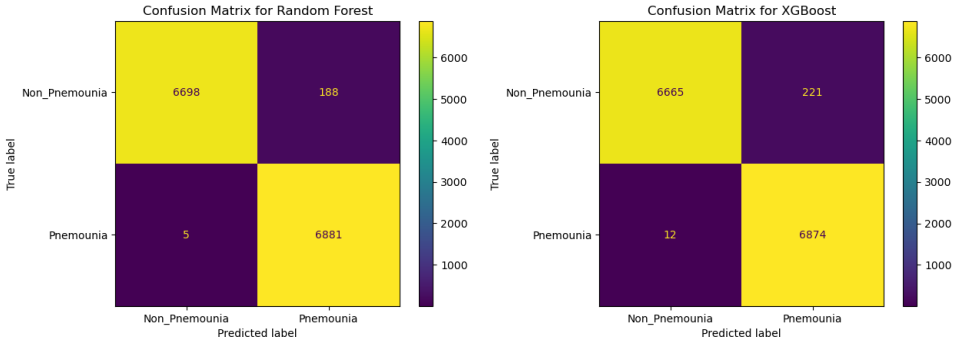

Figure 10: Confusion matrix for RF and XGBoost on the oversampled dataset.

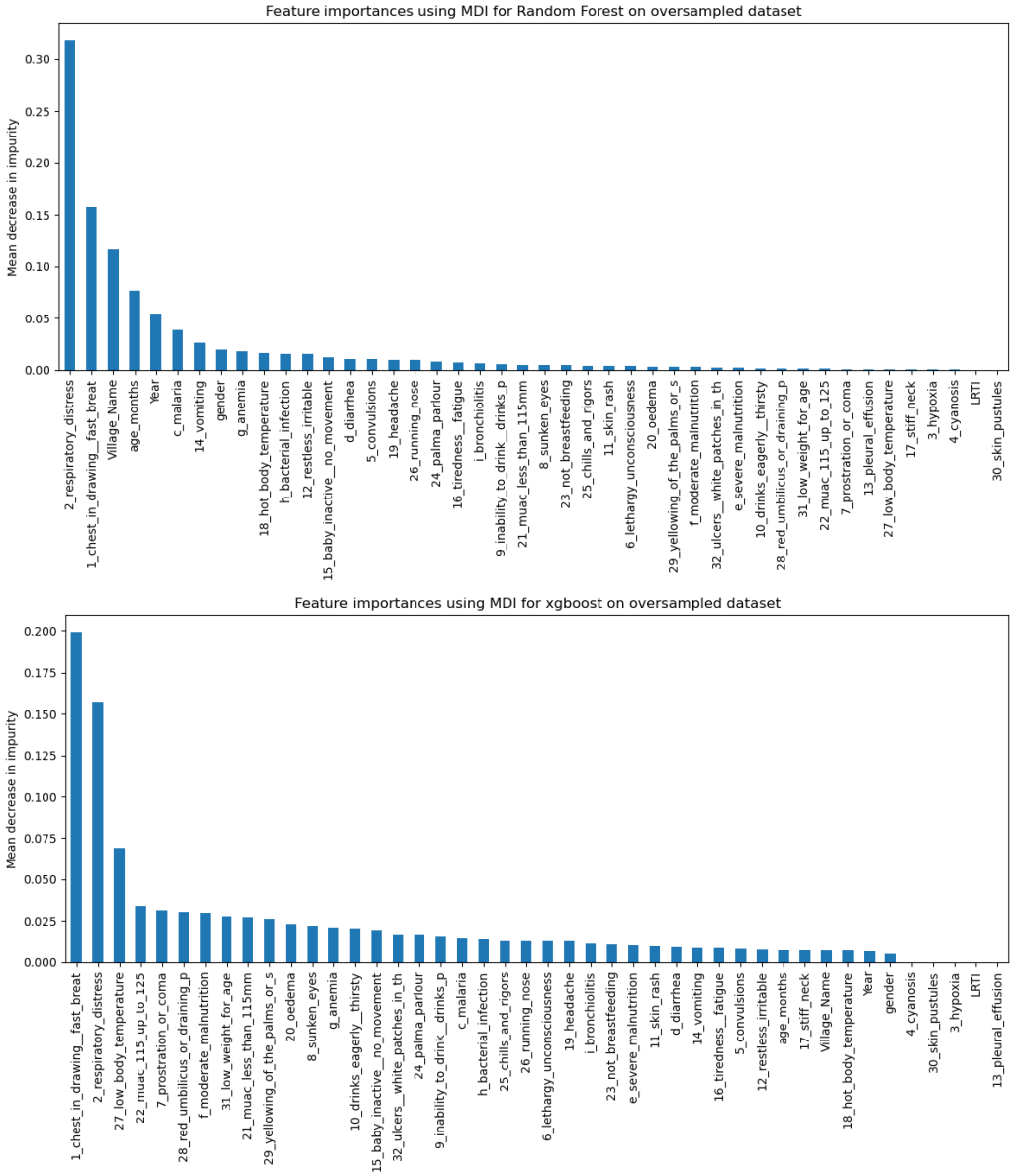

Figure 11: Feature importance of RF and XGboost on the oversampled dataset.

In the SMOTE sampling method, the number of instances for each class was adjusted to 6,886 through the generation of synthetic instances. Table 3 indicates the comparison of six machine learning models on SMOTE datasets. The results show that the XGBoost model had the highest performance compared to the other models. The F1 score for XGBoost was 95%, followed by RF and DT, with scores of 94% and 91%, respectively.

Figure 12 displays that the number of cases incorrectly identified as non-pneumonia (false neg-

atives) for the XGboost model and RF was 303 and 378, respectively. Although,the results showed close performance of both RF and XGBoost models, but XGBoost demonstrated a lower false negative rate compared to RF. This indicates its superior sensitivity in detecting pneumonia cases. Therefore, XGBoost was more effective at minimizing the number of pneumonia cases incorrectly classified as non-pneumonia, which is crucial for ensuring timely and accurate medical interventions.

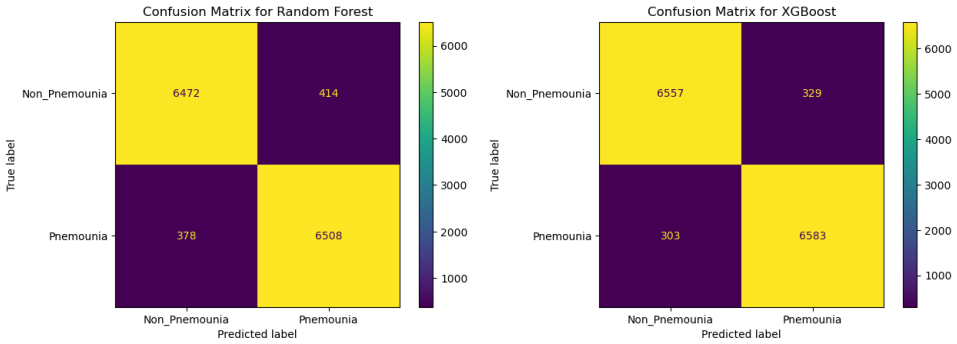

Figure 12: Confusion matrix for RF and XGBoost on the SMOTE dataset.

Feature importance based on the decrease in Gini importance for RF and XGBoost models can be seen in Figure 13. Both the RF and XGBoost models highlighted "respiratory distress" and "chest indrawing with fast breathing" as crucial predictors. These symptoms were consistently identified as essential indicators in diagnosing pneumonia, reflecting their high predictive power across both modeling frameworks. While predictors "bacterial_infection", "malaria" , "anemia", and "diarrhea" identified as most influential for XGBoost. These variables emphasize their strong association with pneumonia. Predictors "Village_name", "c_malaria", "vomiting" and "age" were among the most important variables for Rf model. These predictors underscore that the RF model placed greater emphasis on demographic and symptom indicators.

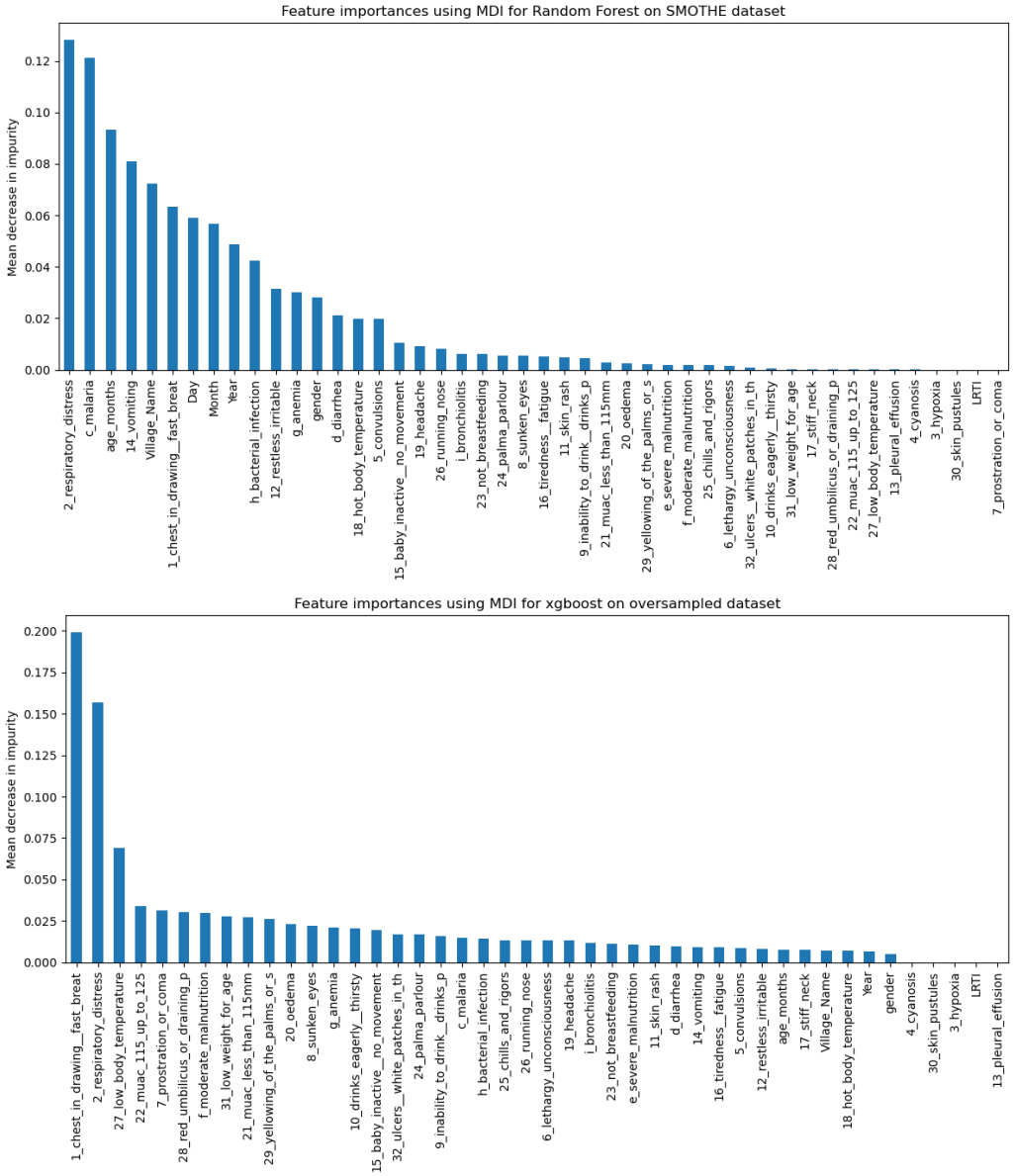

Figure 13: Feature importance of RF and XGBoost on the SMOTE dataset.

Figure14 shows comparison of F1 score of six machine learning models for four distinc datasets. Overall, the results from these three techniques reveal varying levels of effectiveness in addressing class imbalance in the dataset. Firstly, undersampling techniques showed moderate performance across all classifiers, indicating that this method was the least effective compared to the other resampling methods. Undersampling reduces the number of majority class instances to balance the dataset, but it often leads to loss of valuable information and may not adequately represent the over-

all data distribution. Conversely, oversampling and specifically SMOTE demonstrated significant improvements in the RF and XGBoost classifier's performance. By generating synthetic examples of the minority class, SMOTE effectively balanced the dataset and enhanced the classifier's ability to learn from minority class instances. This led to improved predictive accuracy and reduced bias towards the majority class. However, oversampling can increase the potential risk of overfitting due to the duplication of data. Therefore, this method is not recommended.
So, the SMOTE technique among the resampling methods is recommended due to its ability to balance the dataset effectively without as much risk of overfitting. Also, it generates synthetic examples that closely resemble the minority class, thereby enhancing the classifier's ability to generalize to new, unseen data while improving sensitivity to minority class instances.

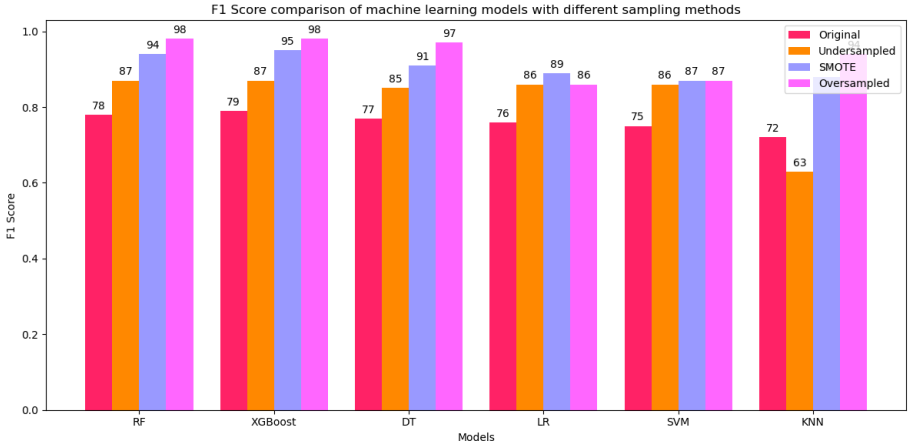

Figure 14: F1 score comparison of machine learning models with different datasets.

5 Possible drawbacks of the used methods

While this study presents promising approaches for predicting pneumonia in children under five using machine learning models, several potential drawbacks and limitations must be considered. Firstly, the study relies on historical clinical data, which may not capture all factors that would play an important role in the diagnosis of pneumonia. The dataset may also contain inherent bias due to its specific geographical and temporal context, which could therefore challenge the generalizability of findings to other regions or time frames.

Moreover, machine learning models including LR, SVM, DT, KNN, RF, and XGBoost, each have their limitations. For example, models like DT and KNN are prone to overfitting, especially in the presence of noisy data. Meanwhile, complex models like RF and XGBoost are powerful but require a lot of computational resources and are also not so interpretable.

Additionally, the resampling techniques utilized in this study to reduce class imbalances like undersampling, oversampling, and SMOTE can introduce their issues. Undersampling removes samples which could lead to the loss of valuable information, while oversampling and SMOTE can cause overfitting by inflating the minority class. Finally, the ethical and practical implications of

these predictive models in real-world healthcare settings need to be carefully considered. Issues such as data privacy, the need for continuous model updating, and the integration of these tools into existing healthcare workflows cause significant challenges.

6 Ethical thinking, societal relevance, and stakeholder awareness

Early detection of pneumonia especially in children under five years old is essential. When pneumonia is recognized at an early stage, timely treatment can be utilized. This early detection can help in reducing the risk of severe complications, such as respiratory failure or secondary infections. So, by using machine learning models, healthcare providers can more effectively recognize children who are at higher risk of developing pneumonia. These models can analyze various factors and predict the most vulnerable children. Additionally, this allows healthcare to prioritize their resources and interventions for those at greatest risk, ensuring that they receive the necessary medical attention and care. This approach can potentially save many lives by preventing the disease from progressing to a critical stage. Moreover, pneumonia prediction through predictive modeling can significantly help in the development of preventive strategies and public health initiatives. Meanwhile, to accurate predictions, healthcare providers can implement community measures such as vaccination campaigns, education on early symptoms, and strategies to reduce exposure to risk factors. These efforts can help in reducing the overall rate of pneumonia, thereby reducing the burden on healthcare systems and families. This approach ensures that the disease is managed not just at the individual level, but also at the community and national levels.

During these few months in order to ensure the study's success, I actively engaged with both internal and external supervisors. We held meetings to explain the progress of the study and gathered their feedback and support. Their input helped design a study that was relevant and respectful to the community's needs. Meanwhile, transparency and honesty are met throughout the study in reporting the research findings. Any limitations, potential biases, and conflicts of interest were reported. The results were presented accurately and without manipulation to reflect the true outcomes of the study.

7 Conclusion

This study predicted pneumonia in children under five years based on symptoms and demographic features. The results of considering only symptoms and demographics revealed that two features " respiratory distress " and "chest indrawing " combined with other features were identified as important predictors for the prediction of pneumonia. Among the balanced datasets, undersampling due to removing information had moderate results. However, the SMOTE resampling method seems to work well, whereas the oversampling method is not recommended due to its vulnerability to overfitting. Therefore, the predictions based on the SMOTE-balanced dataset will be recommended over two other sampling methods for its superior accuracy. Compared to SMOTE dataset, original dataset recommended for prediction of pneumonia in children under five years. Since, the original dataset faithfully reflects the natural distribution of classes encountered in real-world applications. Evaluating models on the original dataset ensures that performance metrics accurately reflect their capabilities in real-world scenarios. So, the original dataset is sufficient to predict pneumonia in children.

Overall, RF, XGBoost, and DT showed better performance with fewer misclassified cases across four different dataset. This strong performance is primarily attributed to the fact that RF and XGBoost are ensemble methods, which combine multiple trees to improve accuracy, while DT operates as a single tree model. Ensemble methods are generally more robust against overfitting, noise, and outliers compared to standalone models like DT. Meanwhile, RF and XGBoost stands out due to its balanced approach of using multiple trees. These method are especially effective in handling imbalanced datasets. As a result, they provide more accurate and reliable predictions compared to other models in both original and balanced datasets. So, the consistent and balanced performance of these two ensemble methods make them particularly effective choice for predicting pneumonia in children under five years.

8 Ideas for future research

In future research, it is recommended to consider additional data sources such as genetic information, economic status, and environmental conditions. This can provide a better comprehensive understanding of important factors that influence the risk of pneumonia and improve the performance of models.

Additionally, more advanced machine learning techniques, like deep learning, could be considered. Also, research should investigate the development and testing of such models and address how these models are interpretable and computationally feasible. Investigation into the effectiveness of various preventive measures, including vaccination programs and public health campaigns, could add evidence to support or contradict such strategies for reducing the incidence of pneumonia.

References

1. Peyrani, P., Mandell, L., Torres, A. & Tillotson, G. The burden ofcommunity-acquired bacterial pneumonia in the era of antibiotic resistance. *Expert Rev Respir Med.* 10.1080/17476348.2019.1562339 (2019).
2. Garber MD.and Quinonez, R. Chest radiograph for childhood pneumonia: good, but not good enough. *American Academy of Pediatrics.* 10.1542/peds.2018-2025 (2018).
3. Deo, R. Machine learning in medicine. *American Academy of Pediatrics.* 10.1161/CIRCULATIONAHA.115.001593 (2015).
4. Beam AL.and Kohane, I. Big data and machine learning in health care. *Jama.* 10.1001/jama.2017.18391 (2018).
5. Naydenova, E., Tsanas, A., Howie, S., Casals-Pascual, C. & De Vos, M. The power of data mining in diagnosis of childhood pneumonia. *J R Soc Interface.* 10.1098/rsif.2016.0266 (2016).
6. Garet, J. & Yunqian, M. *An Introduction into Statistical Learning* (Springer, 2013).
7. Clement, Y., Ruoqi, M., Emmanuel, K., Clement, A. & Ruiping, Q. Machine learning-assisted prediction of pneumonia based on non-invasive measures. *Front. Public Health.* 10.3389/fpubh.2022.938801 (2022).
8. Tarid, W., Surina, H. & Okan, B. A Comparison of Undersampling, Oversampling, and SMOTE Methods for Dealing with Imbalanced Classification in Educational Data Mining. *information.* https://doi.org/10.3390/info14010054 (2023).
9. EMMANUEL, I., YANXIA, S. & ZENGHUI, W. Performance Evaluation of Machine Learning Methods for Credit Card Fraud Detection Using SMOTE and AdaBoost. *IEEE.* 10.1109/ACCESS.2021.3134330 (2021).
10. Jumanah, R. & Malak, A. Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results. *IEEE.* 10.1109/ICICS49469.2020.239556 (2020).

Table 4: Hyperparameter configurations for different datasets

Model	Hyperparameter
	Original Dataset
RF	{'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
XGBoost	{'colsample_bytree': 1.0, 'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 300, 'subsample': 0.6}
DT	{'max_depth': 10, 'max_features': None, 'min_samples_leaf': 5, 'min_samples_split': 20}
LR	{'C': 100, 'penalty': 'l2', 'solver': 'lbfgs'}
SVM	{'kernel' :'poly'}
KNN	{'algorithm': 'kd_tree', 'leaf_size': 20, 'n_neighbors': 3, 'p': 1, 'weights': 'distance'}
	Undersampled Dataset
RF	{'max_depth': 30, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}
XGBoost	{'colsample_bytree': 0.8, 'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 300, 'subsample': 0.6}
DT	max_depth = 10, max_features = None, min_samples_leaf = 10, min_samples_split = 2
LR	{'C': 0.01, 'penalty': 'none', 'solver': 'lbfgs'}
SVM	{'kernel' : 'poly'}
KNN	{'algorithm': 'kd_tree', 'leaf_size': 10, 'n_neighbors': 11, 'p': 1, 'weights': 'distance'}
	Oversampled Dataset
RF	{'max_depth': None, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}
XGBoost	{'colsample_bytree': 0.8, 'learning_rate': 0.2, 'max_depth': 9, 'n_estimators': 300, 'subsample': 0.8}
DT	{'max_depth': 40, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2}
LR	{'C': 0.01, 'max_iter': 100, 'penalty': 'l1', 'solver': 'liblinear'}
SVM	{'kernel':'poly'}
KNN	{'algorithm': 'auto', 'leaf_size': 10, 'n_neighbors': 3, 'p': 1, 'weights': 'distance'}
	SMOTE Dataset
RF	{'max_depth': 30, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
XGBoost	{'colsample_bytree': 0.6, 'learning_rate': 0.2, 'max_depth': 9, 'n_estimators': 300, 'subsample': 0.8}
DT	{'max_depth': 40, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
LR	{'C': 100, 'penalty': 'l2', 'solver': 'liblinear'}
SVM	{'kernel' : 'poly'}
KNN	{'algorithm': 'auto', 'leaf_size': 10, 'n_neighbors': 3, 'p': 1, 'weights': 'distance'}

9 Software Code

All machine learning models:

Desicion Tree:

```
import pandas as pd
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Initialize StratifiedKFold with desired number of folds
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Initialize your classifier (Replace this with your classifier initialization code)
classifier_dt = DecisionTreeClassifier()

# Initialize empty lists to store evaluation metrics for each fold
accuracies = []
precisions = []
recalls = []
f1_scores = []
threshold = 0.5
# Iterate through folds
for train_index, test_index in skf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train your classifier (Replace this with your model training code)
    classifier_dt.fit(X_train, y_train)

    # Evaluate your classifier using various metrics
    y_pred = classifier_dt.predict(X_test)

    binary_dt_pred = [1 if pred >= threshold else 0 for pred in y_pred]
    accuracy = accuracy_score(y_test, binary_dt_pred)
    precision = precision_score(y_test, binary_dt_pred)
    recall = recall_score(y_test, binary_dt_pred)
    f1 = f1_score(y_test, binary_dt_pred)

    # Append metrics to respective lists
    accuracies.append(accuracy)
    precisions.append(precision)
    recalls.append(recall)
    f1_scores.append(f1)
```

```
    # Print metrics for the current fold
    print(f"Fold Accuracy: {accuracy}")
    print(f"Fold Precision: {precision}")
    print(f"Fold Recall: {recall}")
    print(f"Fold F1 Score: {f1}")
    print("\n")

mean_accuracy = sum(accuracies) / len(accuracies)
mean_precision = sum(precisions) / len(precisions)
mean_recall = sum(recalls) / len(recalls)
mean_f1_score = sum(f1_scores) / len(f1_scores)

print(f"Mean Accuracy: {mean_accuracy}")
print(f"Mean Precision: {mean_precision}")
print(f"Mean Recall: {mean_recall}")
print(f"Mean F1 Score: {mean_f1_score}")

Random Forest :
import pandas as pd
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matri


# Initialize StratifiedKFold with desired number of folds
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)


classifier_rf = RandomForestClassifier()

# Initialize empty lists to store evaluation metrics for each fold
accuracies = []
precisions = []
recalls = []
f1_scores = []
num_classes = len(np.unique(y))
aggregated_conf_matrix = np.zeros((num_classes, num_classes), dtype=int)


# Iterate through folds
for train_index, test_index in skf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train your classifier (Replace this with your model training code)
    classifier_rf.fit(X_train, y_train)
```

```
    # Evaluate your classifier using various metrics
    y_pred = classifier_rf.predict(X_test)

    fold_conf_matrix = confusion_matrix(y_test, y_pred, labels=np.unique(y))


    aggregated_conf_matrix += fold_conf_matrix
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    # Append metrics to respective lists
    accuracies.append(accuracy)
    precisions.append(precision)
    recalls.append(recall)
    f1_scores.append(f1)

    # Print metrics for the current fold
    print(f"Fold Accuracy: {accuracy}")
    print(f"Fold Precision: {precision}")
    print(f"Fold Recall: {recall}")
    print(f"Fold F1 Score: {f1}")
    print("\n")

# After the loop, you can perform any post-processing or analysis on the results
# For example, you can calculate mean and standard deviation of the metrics across folds
mean_accuracy = sum(accuracies) / len(accuracies)
mean_precision = sum(precisions) / len(precisions)
mean_recall = sum(recalls) / len(recalls)
mean_f1_score = sum(f1_scores) / len(f1_scores)

print(f"Mean Accuracy: {mean_accuracy}")
print(f"Mean Precision: {mean_precision}")
print(f"Mean Recall: {mean_recall}")
print(f"Mean F1 Score: {mean_f1_score}")


XGBoost :

import pandas as pd
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import  GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
# Initialize StratifiedKFold with desired number of folds
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)


classifier_xgb = GradientBoostingClassifier()

# Initialize empty lists to store evaluation metrics for each fold
accuracies = []
precisions = []
recalls = []
f1_scores = []
num_classes = len(np.unique(y))
aggregated_conf_matrix = np.zeros((num_classes, num_classes), dtype=int)

threshold = 0.5
# Iterate through folds
for train_index, test_index in skf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train your classifier (Replace this with your model training code)
    classifier_xgb.fit(X_train, y_train)

    # Evaluate your classifier using various metrics
    y_pred = classifier_xgb.predict(X_test)
    fold_conf_matrix = confusion_matrix(y_test, y_pred, labels=np.unique(y))


    aggregated_conf_matrix += fold_conf_matrix

    binary_xgb_pred = [1 if pred >= threshold else 0 for pred in y_pred]
    accuracy = accuracy_score(y_test, binary_xgb_pred)
    precision = precision_score(y_test, binary_xgb_pred)
    recall = recall_score(y_test, binary_xgb_pred)
    f1 = f1_score(y_test, binary_xgb_pred)

    # Append metrics to respective lists
    accuracies.append(accuracy)
    precisions.append(precision)
    recalls.append(recall)
    f1_scores.append(f1)

    # Print metrics for the current fold
    print(f"Fold Accuracy: {accuracy}")
    print(f"Fold Precision: {precision}")
```

```
    print(f"Fold Recall: {recall}")
    print(f"Fold F1 Score: {f1}")
    print("\n")

mean_accuracy = sum(accuracies) / len(accuracies)
mean_precision = sum(precisions) / len(precisions)
mean_recall = sum(recalls) / len(recalls)
mean_f1_score = sum(f1_scores) / len(f1_scores)

print(f"Mean Accuracy: {mean_accuracy}")
print(f"Mean Precision: {mean_precision}")
print(f"Mean Recall: {mean_recall}")
print(f"Mean F1 Score: {mean_f1_score}")
```

Logistic Regression:

```
import pandas as pd
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Initialize StratifiedKFold with desired number of folds
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

classifier_lr = LogisticRegression()

# Initialize empty lists to store evaluation metrics for each fold
accuracies = []
precisions = []
recalls = []
f1_scores = []
threshold = 0.5

# Iterate through folds
for train_index, test_index in skf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train your classifier
    classifier_lr.fit(X_train, y_train)

    # Evaluate your classifier using various metrics
    y_pred_proba = classifier_lr.predict(X_test)

    binary_lr_pred = [1 if pred >= threshold else 0 for pred in y_pred_proba]
    accuracy = accuracy_score(y_test, binary_lr_pred)
    precision = precision_score(y_test, binary_lr_pred)
```

```
    recall = recall_score(y_test, binary_lr_pred)
    f1 = f1_score(y_test, binary_lr_pred)

    # Append metrics to respective lists
    accuracies.append(accuracy)
    precisions.append(precision)
    recalls.append(recall)
    f1_scores.append(f1)

    # Print metrics for the current fold
    print(f"Fold Accuracy: {accuracy}")
    print(f"Fold Precision: {precision}")
    print(f"Fold Recall: {recall}")
    print(f"Fold F1 Score: {f1}")
    print("\n")

mean_accuracy = sum(accuracies) / len(accuracies)
mean_precision = sum(precisions) / len(precisions)
mean_recall = sum(recalls) / len(recalls)
mean_f1_score = sum(f1_scores) / len(f1_scores)

print(f"Mean Recall: {mean_recall}")
print(f"Mean F1 Score: {mean_f1_score}")
print(f"Mean Accuracy: {mean_accuracy}")
print(f"Mean Precision: {mean_precision}")

Support vector machine:
import pandas as pd
from sklearn.model_selection import StratifiedKFold
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Initialize StratifiedKFold with desired number of folds
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)


classifier_svc = SVC(kernel='poly')

# Initialize empty lists to store evaluation metrics for each fold
accuracies = []
precisions = []
recalls = []
f1_scores = []
threshold = 0.5

# Iterate through folds
```

```
for train_index, test_index in skf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train your classifier
    classifier_svc.fit(X_train, y_train)

    # Evaluate your classifier using various metrics
    y_pred_proba = classifier_svc.predict(X_test)

    binary_svm_pred = [1 if pred >= threshold else 0 for pred in y_pred_proba]
    accuracy = accuracy_score(y_test, binary_svm_pred)
    precision = precision_score(y_test, binary_svm_pred)
    recall = recall_score(y_test, binary_svm_pred)
    f1 = f1_score(y_test, binary_svm_pred)

    # Append metrics to respective lists
    accuracies.append(accuracy)
    precisions.append(precision)
    recalls.append(recall)
    f1_scores.append(f1)

    # Print metrics for the current fold
    print(f"Fold Accuracy: {accuracy}")
    print(f"Fold Precision: {precision}")
    print(f"Fold Recall: {recall}")
    print(f"Fold F1 Score: {f1}")
    print("\n")

mean_accuracy = sum(accuracies) / len(accuracies)
mean_precision = sum(precisions) / len(precisions)
mean_recall = sum(recalls) / len(recalls)
mean_f1_score = sum(f1_scores) / len(f1_scores)

print(f"Mean Accuracy: {mean_accuracy}")
print(f"Mean Precision: {mean_precision}")
print(f"Mean Recall: {mean_recall}")
print(f"Mean F1 Score: {mean_f1_score}")

KNN:
import pandas as pd
from sklearn.model_selection import StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Initialize StratifiedKFold with desired number of folds
```

```
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

classifier_knn = KNeighborsClassifier()

# Initialize empty lists to store evaluation metrics for each fold
accuracies = []
precisions = []
recalls = []
f1_scores = []
threshold = 0.5

# Iterate through folds
for train_index, test_index in skf.split(X, y):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train your classifier
    classifier_knn.fit(X_train, y_train)

    # Evaluate your classifier using various metrics
    y_pred_proba = classifier_knn.predict(X_test.values)

    binary_knn_pred = [1 if pred >= threshold else 0 for pred in y_pred_proba]
    accuracy = accuracy_score(y_test, binary_knn_pred)
    precision = precision_score(y_test, binary_knn_pred)
    recall = recall_score(y_test, binary_knn_pred)
    f1 = f1_score(y_test, binary_knn_pred)

    # Append metrics to respective lists
    accuracies.append(accuracy)
    precisions.append(precision)
    recalls.append(recall)
    f1_scores.append(f1)

    # Print metrics for the current fold
    print(f"Fold Accuracy: {accuracy}")
    print(f"Fold Precision: {precision}")
    print(f"Fold Recall: {recall}")
    print(f"Fold F1 Score: {f1}")
    print("\n")

mean_accuracy = sum(accuracies) / len(accuracies)
mean_precision = sum(precisions) / len(precisions)
mean_recall = sum(recalls) / len(recalls)
mean_f1_score = sum(f1_scores) / len(f1_scores)
```

```
print(f"Mean Accuracy: {mean_accuracy}")
print(f"Mean Precision: {mean_precision}")
print(f"Mean Recall: {mean_recall}")
print(f"Mean F1 Score: {mean_f1_score}")
%%%%
Display the confusion matrix :
# Display the aggregated confusion matrix using ConfusionMatrixDisplay
disp = ConfusionMatrixDisplay(confusion_matrix=aggregated_conf_matrix, display_labels=["Non_Pnemouni
fig, ax = plt.subplots()  # Set the figure size
disp.plot(ax=ax)
plt.title('Confusion Matrix for Random Forest')
plt.show()
%%%%
Feature Importance:
feature_names = list(X.columns)

forest_importances = pd.Series(classifier_rf.feature_importances_, index=feature_names).sort_values(

fig, ax = plt.subplots(figsize=(12, 7))
forest_importances.plot.bar( ax=ax)
ax.set_title("Feature importances using MDI for Random Forest on undersampled dataset")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()

%%%

% Mehtods fo balancing data set
% Methods1 : Undersampling
count_class_0, count_class_1 = new_sympthoms.pneumonia.value_counts()

# Divide by class
df_class_0 = new_sympthoms[new_sympthoms['pneumonia'] == 0]
df_class_1 = new_sympthoms[new_sympthoms['pneumonia'] == 1]
df_class_0_under = df_class_0.sample(count_class_1)
df_test_under = pd.concat([df_class_0_under, df_class_1], axis=0)

print('Random under-sampling:')
print(df_test_under.pneumonia.value_counts())
X = df_test_under.drop(['pneumonia'], axis=1)
y = df_test_under['pneumonia']

%%% Method2 : Oversampling methods

# Oversample 1-class and concat the DataFrames of both classes
df_class_1_over = df_class_1.sample(count_class_0, replace=True)
df_test_over = pd.concat([df_class_0, df_class_1_over], axis=0)
```

```
print('Random over-sampling:')
print(df_test_over.pneumonia.value_counts())
X = df_test_over.drop(['pneumonia'], axis=1)
y = df_test_over['pneumonia']

%%% Method3 : SMOTE

X_1 = new_sympthoms.drop(['pneumonia'], axis=1)
y_1 = new_sympthoms['pneumonia']
from imblearn.over_sampling import SMOTE


smote = SMOTE(sampling_strategy='minority')
X, y = smote.fit_resample(X_1, y_1)

y.value_counts()
```