Predicting the Optimal Machine Learning Model for National Poll on Healthy Aging (NPHA) Dataset

Simindokht Mobasheri
Dept. Computer Engineering
Sharif University of Technology
Kish, Iran
Email: simindokhtmobasheri5005@gmail.com

Maryam Chamkouri
Dept. Computer Engineering
Sharif University of Technology
Kish, Iran
Email: maryamchamkouri@gmail.com

***Abstract*— With the growth of the older population, it is increasingly important to address the health needs of older adults. The National Poll on Healthy Aging (NPHA) dataset developed by the University of Michigan is a valuable resource for information about the health concerns of Americans aged 50 and older. This study predicts health risk through analysis of a subset of the NPHA dataset with 14 sleep and health features. Machine learning techniques like feature engineering and hyperparameter optimization, to improve prediction accuracy. Our findings suggest that feature engineering helps to enhance model performance, showing that these models can give reliable insights for predicting health risk among older adults. This research emphasizes the utility of data-driven approaches to inform healthcare policy and improve interventions among elderly populations.**

Index Terms—aging, health outcomes, machine learning, feature engineering, predictive modeling.

***Index Terms*—**

I. Introduction

The aging population poses significant challenges to healthcare systems, with older adults facing increasing risks of chronic conditions like cardiovascular disease, diabetes, and cognitive decline. Addressing these health needs is a priority for public health initiatives. The National Poll on Healthy Aging (NPHA) dataset, curated by the University of Michigan, provides valuable insights into the health issues of Americans aged 50 and older, including sleep patterns, medication use, and chronic conditions. This study aims to use machine learning to predict health risks based on 14 selected features from the NPHA dataset, helping to identify at-risk individuals and enhance healthcare strategies.

The study investigates several machine learning algorithms chosen for their effectiveness in classification tasks, including Logistic Regression, a probabilistic model using a sigmoid function; Decision Trees, which partition data recursively; Random Forest, an ensemble method combining multiple trees; AdaBoost, which adjusts the weight of misclassified instances; Gaussian Naive Bayes, assuming Gaussian feature distributions; Support Vector Machines (SVM), which find the optimal hyperplane for separation; and K-Nearest Neighbors (KNN), classifying based on majority voting among neighbors. Several challenges were encountered throughout the study. One key challenge was feature engineering, where identifying and selecting the most impactful features from the raw dataset was crucial for improving model accuracy. Feature combination and interaction also posed difficulties, as it was important to explore both linear and non-linear relationships between features to optimize predictive performance. Moreover, finding the best combination of techniques—ranging from feature selection to model optimization—was a significant challenge in determining which methods would lead to the best possible model. Additionally, addressing the class imbalance in the dataset was important to ensure that the model could effectively predict risks for both majority and minority classes. Fine-tuning the hyper-parameters for each model was necessary to ensure the best performance across various metrics, including accuracy, precision, recall, and F1-score. This study employed several techniques to improve model performance and predict health risks in older adults. Feature engineering included selecting relevant features using RFE and SFS, while evaluating Feature Importance and Permutation Importance. Feature combinations were optimized using Cross-Validation, GridSearch, and RFE. Hyper-parameter tuning was performed with GridSearchCV, Random Search, and Bayesian Optimization. PCA was investigated for dimensionality reduction to capture the maximum variance from the data with fewer features. Additionally, Data Augmentation techniques were explored to address class imbalance and improve the model's ability to generalize, with their effectiveness evaluated to determine if they resulted in performance improvements before being incorporated into the final models. K-Fold and Stratified K-Fold cross-validation ensured reliable performance estimates and prevented overfitting. The models' performance was evaluated across multiple metrics to ensure accuracy in predicting health risks. best model: KNN, Adaboost

The structure of the paper is as follows: Section II provides an in-depth exploratory data analysis (EDA) of (NPHA) dataset, covering data cleaning, feature analysis, and visualizations. Section III outlines the machine learning models used, along with the data preprocessing methods and their mathematical formulations. In Section IV, a comparison of the model's performance is presented, based on the selected evaluation metrics. Finally, Section V concludes the paper by summarizing the results, addressing limitations, and proposing directions for future research.

II. Exploratory Data Analysis (EDA)

We begin by performing EDA on the dataset. The dataset consists of:

- Overview of the Dataset

A. Description:

The dataset is a branch from NPHA and had been contributed on 5th December 2023. It had been designed especially for testing predictive machine-learning algorithms aimed at predicting doctors' annual consultations with individual respondents. This dataset now represents the elderly population aged 50 and above who participated in the survey with NPHA.

B. Characteristics:

- Type: Tabular
- Subject Area: Health and Medicine
- Associated Tasks: Classification
- Number of Instances: 714
- Number of Features: 14

C. Data Preprocessing:

For this specific subset of the original NPHA data set, 14 health and sleep features were selected for prediction work. Keeping the integrity of the database in check, where the chosen features have missing responses from survey respondents, these entries were removed from the database.

D. Feature Description:

1) Class Labels:: The target variable for this dataset is the Number of Doctors Visited, and respondents are categorized by the total count of different types of doctors they have seen in a year. The class labels are defined as follows:

- 1: 0-1 doctors
- 2: 2-3 doctors
- 3: 4 or more doctors

This categorization will help in predicting healthcare utilization among the elderly based on other health-related characteristics.

2) Feature Descriptions:: The dataset contains 14 features, each providing insights into the health and lifestyle of the respondents. Below is a detailed description of each feature:

- Age:
  - Represents the patient's age group.
  - Values:
    - 1: 50-64
    - 2: 65-80
- Physical Health:
  - A self-assessment of the patient's physical well-being.
  - Values:
    - -1: Refused
    - 1: Excellent
    - 2: Very Good
    - 3: Good
    - 4: Fair
    - 5: Poor
- Mental Health:
  - A self-evaluation of the patient's mental or psychological health.
  - Values:
    - -1: Refused
    - 1: Excellent
    - 2: Very Good
    - 3: Good
    - 4: Fair
    - 5: Poor
- Dental Health:
  - A self-assessment of the patient's oral or dental health.
  - Values:
    - -1: Refused
    - 1: Excellent
    - 2: Very Good
    - 3: Good
    - 4: Fair
    - 5: Poor
- Employment:
  - The patient's employment status or work-related information.
  - Values:
    - -1: Refused
    - 1: Working full-time
    - 2: Working part-time
    - 3: Retired
    - 4: Not working at this time
- Stress Keeps Patient from Sleeping:
  - Indicates whether stress affects the patient's ability to sleep.
  - Values:
    - 0: No
    - 1: Yes
- Medication Keeps Patient from Sleeping:
  - Indicates whether medication impacts the patient's sleep.
  - Values:
    - 0: No
    - 1: Yes
- Pain Keeps Patient from Sleeping:
  - Indicates whether physical pain disturbs the patient's sleep.
  - Values:
    - 0: No
    - 1: Yes
- Bathroom Needs Keeps Patient from Sleeping:

    - Indicates whether the need to use the bathroom affects the patient's sleep.
    - Values:
        - 0: No
        - 1: Yes
- Unknown Keeps Patient from Sleeping:
    - Indicates unidentified factors affecting the patient's sleep.
    - Values:
        - 0: No
        - 1: Yes
- Trouble Sleeping:
    - General issues or difficulties the patient faces with sleeping.
    - Values:
        - 0: No
        - 1: Yes
- Prescription Sleep Medication:
    - Information about any sleep medication prescribed to the patient.
    - Values:
        - -1: Refused
        - 1: Use regularly
        - 2: Use occasionally
        - 3: Do not use
- Race:
    - The patient's racial or ethnic background.
    - Values:
        - -2: Not asked
        - -1: Refused
        - 1: White, Non-Hispanic
        - 2: Black, Non-Hispanic
        - 3: Other, Non-Hispanic
        - 4: Hispanic
        - 5: 2+ Races, Non-Hispanic
- Gender:
    - The gender identity of the patient.
    - Values:
        - -2: Not asked
        - -1: Refused
        - 1: Male
        - 2: Female

III. Handling Missing Values

The entries with missing responses are removed before constructing the subset dataset. However, coded values such as -1 (refused) and -2 (not asked) represent a form of missing data that needs special attention.

A. Replace with NaN:

Replace coded missing values (e.g., -1, -2) with NaN (Not a Number).

B. Check for Undefined Values:

Check that the dataset does not have values other than what is defined for us. If there are, we will convert them to NaN as well.

C. Handle NaN:

- Impute: Impute the coded missing values with a calculated value.
    - Using mode: The mode is one measure of central tendency that indicates the point that occurs most frequently in a dataset. Unlike the mean or average of a set of values or the median, which states the middle position in an ordered set, mode is tremendously useful with categorical data when you wish to identify the most popular category or value.
    - Ex. Mental_Health
- Drop: Remove Missing Values
    - If the number of missing values is small compared to the dataset size.
    - Ex. Trouble_Sleeping

IV. Explore Data Characteristics

A. Univariate Analysis:

Focuses on studying one variable to understand characteristics of data, i.e., describe the data and find patterns within a single feature.

B. Bivariate Analysis:

The relationships between two or more variables in the dataset to furnish connections, correlations, and dependencies.

C. Correlation Analysis:

Analyzes the correlation between some of the dataset features with a specific focus on establishing interactions between them.

V. Feature Engineering

For the purpose of enhancing the power of our model, two composite features were built: Overall Health Score and Sleep Disturbance Score.

A. Composite Health Score:

To quantify a patient's overall health status, we combined 3-health-related features: Physical_Health, Mental_Health, and Dental_Health. Each of these features was normalized within a range of [0, 1] by dividing them by the maximum value for that feature. This normalization allows the feature to contribute equally toward the composite score, no matter what the original scale is like. The normalized features were summed to give the Overall_Health feature or a cumulative indication of the health status for each patient. The original health-related features are dropped to avoid redundancy and reduce multicollinearity.

B. Sleep Disturbance Score:

Sleep disturbances are a significant factor for patient health and well-being. In order to obtain the overall impact of various factors of sleep disturbances, we combined five binary features: Stress_Keeps_Patient_from_Sleeping, Medication_Keeps_Patient_from_Sleeping, Pain_Keeps_Patient_from_Sleeping, Bathroom_Needs_Keeps_Patient_from_Sleeping, and Unknown_Keeps_Patient_from_Sleeping. Each feature indicates whether a specific factor keeps the patient awake or not. Adding these binary features, we obtained a Sleep_Disturbance score, which is the number of total amounts of disturbances of sleep for a patient. The original sleep-related features were dropped to reduce the dataset and remove noise. The Overall Health feature was generated by normalizing and summing the Physical Health, Mental Health, and Dental Health features, giving a sort of comprehensive measure of patient health. The Sleep Disturbance feature was calculated by summing five binary sleep-related features, capturing the overall effect of sleep disturbances directly. These engineered variables provided a dual benefit of dimensionality reduction of the data and enhanced model interpretability and performance.

VI. Data Normalization/Scaling

At the end of the preprocessing process, we have two datasets, one with 11 features and the other with 7. One set of these features is binary.

- Binary Features: Binary features (0 or 1) do not require normalization because they are already in a normalized scale. Having only two values, scaling them will not change their representation or the model's performance.
- Limited Range of Discrete Features: The discrete features, whose values lie between 1 and 4 or 5, are already in a small range. Normalization generally attempts to bring features to the same scale, but since these features are already in a small range, scaling is not that required. The values are so close to one another that they can be interpreted without introducing bias.
- New Feature: The last set is a new feature that was created in the feature engineering process and was normalized.
  - Ex. Composite Health Score

VII. Model Discussion

This section describes the implementation and evaluation of three machine learning models: Support Vector Machine (SVM), Logistic Regression, Naive Bayes, KNN, AdaBoost, Decision Tree, and Random Forest on the preprocessed data for predicting doctor visits. We describe the rationale for choosing these models, the specific implementations, hyperparameter tuning strategies, and evaluation metrics used.

A. Support Vector Machine (SVM)

SVM is a strong supervised learning algorithm for identifying the optimal hyperplane which best separates points of different classes (Cortes & Vapnik, 1995) [1]. Its strength is its ability to handle high-dimensional spaces and its ability to use any kernel function to identify non-linear relationships.

1) Implementation: We used SVM with the scikit-learn library (Pedregosa et al., 2011) [2] in Python. Because of the ordinality of some features, we experimented with different kernel functions, including linear, polynomial (of different degrees), and radial basis function (RBF). Model performance as a function of regularization parameter C was also investigated.

2) Hyperparameter Tuning: The optimal hyperparameters were determined by employing a combination of techniques:

- **Grid Search**: Complete grid search over a subset of values of C and kernel parameters (e.g., polynomial degree, RBF gamma).
- **Cross-Validation**: 5-fold stratified cross-validation during the grid search to accurately measure model performance for each hyperparameter interaction and prevent overfitting (Hastie et al., 2009) [3]. Stratified cross-validation maintains the class distribution within the folds, which is necessary for imbalanced classes.
- **Scoring Metric**: The primary metric for determining performance during hyperparameter tuning was the f1-macro score. This score is particularly beneficial for classification tasks with imbalanced classes, as it evaluates performance on each class individually. We also employed other performance metrics like precision, recall, F1-score, and AUC-ROC to learn more about the model's behavior.

The best-performing model based on cross-validation outcomes was selected for final evaluation.

3) Results and Analysis: The performance of the tuned SVM model is presented in Table 1. We analyze the confusion matrix to learn about the model's classification behavior for all classes. We also discuss the impact of varying kernel operations and hyperparameter configurations on the final performance. We inspect the feature importance to identify features with the most influence in the SVM model's prediction.

TABLE I
SVM Performance Metrics

Metric	Mean	Standard Deviation (Std)
Accuracy	0.5077	0.0323
Precision	0.4390	0.0588
Recall	0.5077	0.0323
F1-Score	0.4168	0.0502

B. Logistic Regression

Logistic Regression is a linear binary and multi-class classification classifier (Hosmer & Lemeshow, 2000) [4]. It

estimates the probability of a point belonging to a particular class using a logistic function:

$$P(y = 1|x) = \frac{1}{1 + e^{-(\theta^T x + b)}}$$

where x is the input features, θ are the coefficients, and b is the bias. Not linear in nature, but a good baseline and having feature importance interpretation makes it a strong contender.

1) Implementation: We used Logistic Regression with scikit-learn (Pedregosa et al., 2011). The model was initialized with a maximum of 4000 iterations to ensure convergence. The implementation involved experimenting with different solvers, regularization strengths, and tolerance values to optimize performance.

2) Hyperparameter Tuning: We employed a grid search with 5-fold stratified cross-validation to identify the optimal hyperparameters. The grid search explored combinations of the following:

- **Regularization Strength** (C): Values ranging from 0.001 to 100.
- **Solver**: Various solvers including 'newton-cg', 'lbfgs', 'sag', 'saga', and 'liblinear'.
- **Tolerance** (tol): Values of $1e-4$, $1e-3$, and $1e-2$.
- **Class Weight**: Options of `None` (no weighting) and `'balanced'` (automatic adjustment based on class frequencies).

The primary scoring metric used during hyperparameter tuning was balanced accuracy, which is particularly useful for imbalanced datasets.

3) Results and Analysis: Results for the tuned Logistic Regression model are presented in Table 2. We analyze the logistic regression model coefficients to identify the interaction between features and the target variable. We discuss the impact of different regularization techniques on the coefficients and accuracy of the model.

TABLE II
LOGISTIC REGRESSION PERFORMANCE METRICS

Metric	Mean	Standard Deviation (Std)
Accuracy	0.5138	0.0305
Precision	0.4296	0.0451
Recall	0.5138	0.0305
F1-Score	0.4097	0.0445

VIII. RESULTS AND ANALYSIS

A. ROC Curve Analysis

To compare the performance of the Logistic Regression model, we plotted the Receiver Operating Characteristic (ROC) curve for each class. The ROC curve indicates the trade-off between the True Positive Rate (TPR) and the False Positive Rate (FPR) at various threshold values. The Area Under the Curve (AUC) provides a single value to measure the ability of the model to discriminate between classes.

As shown in Figure 1, the ROC curves for the three classes are as follows:

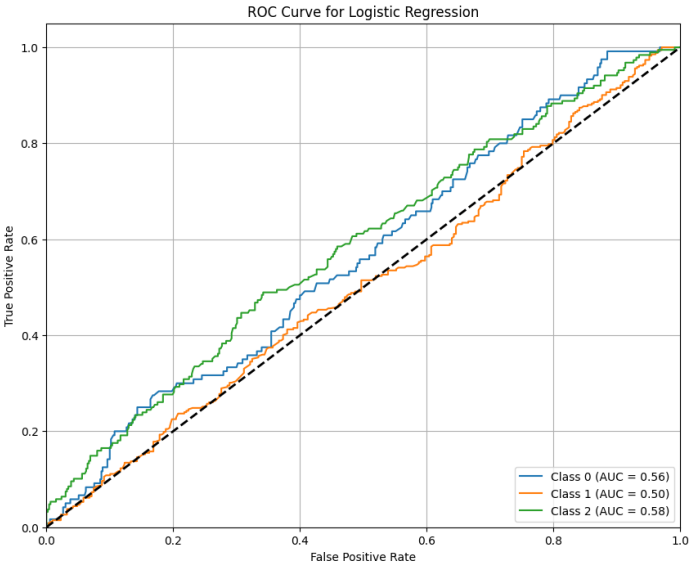

Fig. 1. ROC Curve for Logistic Regression

- **Class 0**: AUC = 0.56
- **Class 1**: AUC = 0.50
- **Class 2**: AUC = 0.58

B. Interpretation

The AUC values indicate that the Logistic Regression model has limited discriminative power:

- **Class 0** (AUC = 0.56) and **Class 2** (AUC = 0.58) a bit more accurate than random chance, but the model is not yet capable of distinguishing these classes correctly.
- **Class 1** (AUC = 0.50) performs no better than random guess, indicating that the model cannot discriminate this class from the other classes.

The crossing curves and low AUC values suggest that the Logistic Regression model may not be a suitable model for this multi-class classification task. Subsequent studies may include more advanced models, e.g., Random Forests or Gradient Boosting, to improve classification accuracy.

C. Naive Bayes

Naive Bayes is a Bayes' theorem-based probabilistic classifier with the "naive" independence assumption of features (Bishop, 2006). Its simplifying assumption despite, it performs remarkably well in the majority of situations, especially in high-dimensional space, and is computationally cheap. Bayes' theorem is given by:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

where y is the class label and x is the feature vector.

1) Implementation: We applied Multinomial Naive Bayes, which is suitable for discrete data, using scikit-learn (Pedregosa et al., 2011). The implementation involved tuning the smoothing parameter (α) and the prior probability of the classes (fit_prior).

2) Tuning Hyperparameter: For Naive Bayes, the only hyperparameter generally to be tuned is smoothing. We used cross-validation to identify the optimal smoothing value. We employed a grid search with 5-fold cross-validation to identify the optimal hyperparameters. The grid search explored combinations of the following:

- **Smoothing Parameter** (α): Values of 0.1, 0.5, 1.0, and 2.0.
- **Fit Prior** (fit_prior): Options of `True` (use the data to estimate class prior probabilities) and `False` (use a uniform prior).

The primary scoring metric used during hyperparameter tuning was accuracy.

3) Results and Analysis: The result of the chosen Naive Bayes model is shown in Table 3. We describe the model performance and look at the class probabilities. We also state whether the feature independence assumption is valid for this data and its potential impact on the model's performance.

TABLE III
NAIVE BAYES PERFORMANCE METRICS

Metric	Mean	Standard Deviation (Std)
Accuracy	0.5292	0.0393
Precision	0.3868	0.1618
Recall	0.5292	0.0393
F1-Score	0.3703	0.0482

D. Model 1: [K-Nearest Neighbors (KNN) Algorithm]

[The K-Nearest Neighbors (KNN) algorithm is a non-parametric method used for classification and regression tasks. It works by identifying the 'k' nearest neighbors to a given data point and making predictions based on the majority class of these neighbors in classification tasks. KNN is simple yet effective, and it does not assume any underlying distribution of the data.]. The mathematical formulation is as follows:

E. Mathematical Formulation

To classify a new data point x, the algorithm calculates the distance between x and all other points in the training set. The two most common distance metrics are Euclidean distance and Manhattan distance, which are defined as follows:

$$d_{\text{Euclidean}}(x, x_i) = \sqrt{\sum_{j=1}^{n} (x_j - x_{ij})^2} \tag{1}$$

$$d_{\text{Manhattan}}(x, x_i) = \sum_{j=1}^{n} |x_j - x_{ij}| \tag{2}$$

Where:

- x is the new data point,
- x_i is the i-th training data point,
- n is the number of features.

The class label of the new data point is assigned based on the majority class among the K nearest neighbors, according to the chosen distance metric.

IX. K-NEAREST NEIGHBORS (KNN)

K-Nearest Neighbors (KNN) is a non-parametric, lazy learning algorithm used for classification tasks. It works by assigning a class to a new data point based on the majority class of its K nearest neighbors. This model is simple and effective, particularly in low-dimensional feature spaces.

A. Implementation

We used the KNN classifier from the scikit-learn library (Pedregosa et al., 2011) to implement the model. The model uses the following parameters:

- **Number of Neighbors** (K): The number of nearest neighbors to consider for classifying a new data point.
- **Distance Metric**: We tested both Euclidean and Manhattan distance metrics.
- **Weights**: We used 'uniform' and 'distance' for weighting neighbors.

B. Feature Selection

Before training the KNN model, feature selection was performed using two methods:

- **Random Forest Feature Importance**: This technique assessed the importance of each feature, and the top 10 features were selected.
- **Permutation Importance**: This method was used to validate and refine the feature selection, identifying the most influential features for model predictions.

The final set of selected features included 'DentalHealth', 'MentalHealth', 'PhysicalHealth', and 'TroubleSleeping', among others.

C. Hyperparameter Tuning

We tuned the KNN model's hyperparameters using GridSearchCV with a 5-fold stratified cross-validation:

- **Grid Search**: We searched over a range of hyperparameters:
  - K: Number of neighbors, tested with values 3, 5, and 7.
  - **Weights**: 'uniform' and 'distance'.
  - **Metric**: 'euclidean' and 'manhattan'.
- **Cross-Validation**: We used 5-fold stratified cross-validation to avoid overfitting and ensure balanced class distributions.
- **Scoring Metric**: We used accuracy, precision, recall, and F1-score as performance metrics during hyperparameter tuning.

D. Results and Analysis

The results of the KNN model are summarized in Table IV. The model's performance was evaluated across training, validation, and test sets, and the ROC curve is shown in Figure **??**.

TABLE IV
PERFORMANCE METRICS FOR KNN MODEL

Metric	Train	Validation	Test
Accuracy	0.59	0.53	0.52
Precision	0.58	0.50	0.49
Recall	0.59	0.53	0.52
F1-Score	0.56	0.49	0.49

E. Discussion

The KNN model showed decent performance with an accuracy of 59% on the training set and 52% on the test set. The best hyperparameters found were $K = 7$, Euclidean distance, and uniform weights. The ROC curve indicated that the model is able to distinguish between health risk classes, but further improvements in hyperparameter tuning or model selection could enhance performance.

F. Conclusion

The KNN algorithm performed reasonably well, but its performance on the test set was lower than on the training set, possibly due to class imbalance or sensitivity to the number of neighbors. Further optimization and exploration of more advanced models, such as Random Forest or Gradient Boosting, may help to improve accuracy and generalization to unseen data.

Next Steps:

- Explore tree algorithm such a classifiers
- Address class imbalance more effectively, using techniques like **SMOTE**.
- Experiment with additional hyperparameters and distance metrics to fine-tune the KNN model.

G. [Decision Tree Algorithm]

The Decision Tree (DT) algorithm is a supervised learning method used for classification tasks. It splits the data at each node, choosing the best feature and threshold to separate the data into homogeneous classes. It works by constructing a tree-like structure where each node represents a decision based on a feature, and each leaf node represents a class label.

H. Mathematical Formulation

A Decision Tree uses the following criteria to decide how to split the data at each node:

1. Gini Impurity is a measure of the impurity or purity of a node. It is calculated as:

$$Gini(t) = 1 - \sum_{i=1}^{k} p_i^2$$

where p_i is the probability of class i in node t, and k is the number of classes.

2. Entropy is another criterion used for splitting the data. It is defined as:

$$Entropy(t) = -\sum_{i=1}^{k} p_i \log_2(p_i)$$

where p_i is the probability of class i in node t.

At each node, the decision tree chooses the feature and threshold that minimizes Gini Impurity or Entropy and maximizes information gain.

I. Implementation

We implemented the Decision Tree model using the scikit-learn library. The model was trained with the following parameters:

- **Criterion**: We tested both Gini Impurity and Entropy as the splitting criteria.
- **Max Depth**: We experimented with various depths to prevent overfitting.
- **Min Samples Split**: The minimum number of samples required to split a node.
- **Min Samples Leaf**: The minimum number of samples required to be at a leaf node.

J. Feature Selection

Feature selection for the Decision Tree model was performed using the following methods:

- **Random Forest Feature Importance**: We used Random Forest to identify the most important features for the Decision Tree model. The top 10 features were selected.
- **Permutation Importance**: This method assessed the impact of each feature on model performance by randomly shuffling features and observing the impact on accuracy.

Selected features for the final model included DentalHealth, MentalHealth, PhysicalHealth, and TroubleSleeping.

K. Hyperparameter Tuning

We tuned the Decision Tree model's hyperparameters using GridSearchCV with 5-fold stratified cross-validation:

- **Grid Search**: We explored the following hyperparameters:
  - Criterion: 'gini' and 'entropy'.
  - Max Depth: [None, 10, 20, 25].
  - Min Samples Split: [2, 5, 20, 30].
  - Min Samples Leaf: [1, 2, 4].
- **Cross-Validation**: We used 5-fold stratified cross-validation to ensure balanced class distributions and avoid overfitting.
- **Scoring Metric**: We used accuracy, precision, recall, and F1-score during hyperparameter tuning.

L. Results and Analysis

The results of the Decision Tree model are summarized in Table VI. The model's performance was evaluated across training, validation, and test sets, and the ROC curve is shown in Figure 2.

TABLE V
PERFORMANCE METRICS FOR DECISION TREE MODEL

Metric	Train	Validation	Test
Accuracy	0.62	0.53	0.46
Precision	0.64	0.58	0.50
Recall	0.62	0.60	0.53
F1-Score	0.57	0.49	0.42

M. Discussion

The Decision Tree model performed well on the training dataset, with an accuracy of 62%, but the performance dropped on the test dataset, which was 46%. The best hyperparameters found were: - Criterion: 'entropy' - Max Depth: 10 - Min Samples Split: 2 - Min Samples Leaf: 4

The ROC curve indicated that the model was able to distinguish between the health risk classes, but further adjustments in hyperparameters or model selection might improve the test set performance.

N. Conclusion

The Decision Tree algorithm performed adequately but showed a noticeable performance drop on the test set. The overfitting issue may be addressed with more advanced models, such as Random Forest or Gradient Boosting. The best hyperparameters were Max Depth = 10, MinSamplesSplit = 2, and Criterion = 'entropy'.

Next Steps:

- Explore more advanced models such as Random Forest
- Experiment with pruning strategies and further hyperparameter tuning to prevent overfitting.

TABLE VI
PERFORMANCE METRICS FOR DECISION TREE MODEL

Metric	Train	Test
Accuracy	62%	46%
Precision	64%	50%
Recall	62%	53%
F1-Score	57%	42%

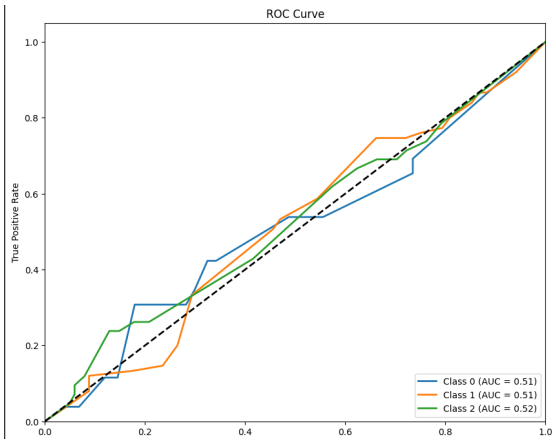

Fig. 2. ROC Curve for Decision Tree Model

X. COMPARISON

We compare the performance of the models based on [metrics]. Table VII summarizes the results. The best-performing model is [model name].

TABLE VII
COMPARISON OF MODELS

Model	Accuracy	Precision	Recall	F1-Score
Model 1	0.XX	0.XX	0.XX	0.XX
Model 2	0.XX	0.XX	0.XX	0.XX
Model 3	0.XX	0.XX	0.XX	0.XX

A. Model 2: AdaBoost Classifier

1) Overview: AdaBoost, short for Adaptive Boosting, is an ensemble learning method that combines multiple weak classifiers to form a stronger classifier. In this study, we used AdaBoostClassifier with a DecisionTreeClassifier as the base estimator to predict health risks in older adults.

2) Mathematical Formulation: AdaBoost works by combining multiple weak classifiers (typically decision trees) to create a strong classifier. It adjusts the weight of each sample in the dataset based on the previous classifier's mistakes. For each weak classifier, AdaBoost computes a weight α_t for the classifier $h_t(x)$ at iteration t as:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

Where ϵ_t is the weighted error rate of classifier $h_t(x)$. The final model prediction is a weighted sum of the predictions of all the weak classifiers:

$$H(x) = \text{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$$

Where T is the total number of weak classifiers and $h_t(x)$ is the prediction of the t-th classifier.

3) Hyperparameter Tuning: We performed hyperparameter tuning for the AdaBoost model using GridSearchCV. The hyperparameters tuned include:

- **Base Estimator**: A Decision Tree Classifier with various configurations (e.g., maxdepth, minsamples split, minsamplesleaf).
- **Number of Estimators**: Number of boosting rounds.
- **Learning Rate**: Controls the contribution of each estimator.

We used Stratified K-Fold Cross-Validation with 5 folds to ensure balanced class distributions and avoid overfitting.

4) Results and Analysis: The results of the AdaBoost model are summarized in Table VIII. The model's performance was evaluated across training, validation, and test sets, and the ROC curve is shown in Figure **??**.

TABLE VIII
PERFORMANCE METRICS FOR ADABOOST MODEL

Metric	Train	Validation	Test
Accuracy	0.79	0.43	0.43
Precision	0.80	0.41	0.41
Recall	0.79	0.43	0.43
F1-Score	0.79	0.42	0.42

5) Discussion: The AdaBoost model showed good performance on the training set with an accuracy of 79%, but its performance on the test set was lower (43% accuracy). The best hyperparameters found were $n_{\text{estimators}} = 50$, learning rate $= 0.1$, and **DecisionTreeClassifier as the base estimator with max depth $= 3$, minsamplessplit = 10, and min samples leaf = 5.

The ROC curve demonstrates the model's ability to distinguish between health risk classes on the training and test data, but the lower test performance suggests the model may be overfitting.

6) Conclusion: The AdaBoost model performed well during training but showed a significant drop in performance on the test set, likely due to class imbalance or hyperparameter sensitivity. The model was optimized with hyperparameters that improved performance on the training data. To further improve the performance, SMOTE or more advanced techniques, such as **Gradient Boosting, may be explored.

Next Steps:

- Explore additional classifiers random forest

XI. CONCLUSION

In this study, we explored the use of machine learning models to predict healthcare utilization among older adults based on the National Poll on Healthy Aging (NPHA) dataset. Through the analysis, we gained some valuable insights into the dataset and the performance of various models.

A. Non-Linear Separability of Classes

In order to better visualize the structure of the dataset, we projected the feature space onto two principal components using Principal Component Analysis (PCA). This allowed us to plot the distribution of the target variable, *Number of Doctors Visited*, in a 2D space. We also trained a Decision Tree classifier on the PCA-transformed data and plotted its decision boundaries.

As shown in Figure 3, the boundary lines drawn by the Decision Tree classifier demonstrate the **non-linear separability** of the classes. The areas where the classes overlap in the graph indicate that the classes (0-1 doctors, 2-3 doctors, and 4 or more doctors) are not linearly separable. This fact confirms the application of non-linear models, e.g., Decision Trees, Random Forests, or Support Vector Machines with non-linear kernels, to effectively represent the complex relationships in the data.

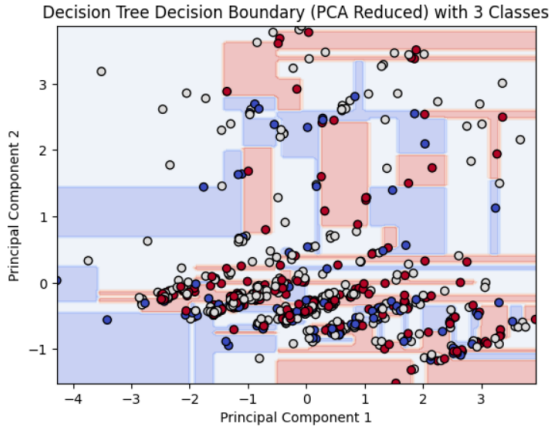

Fig. 3. Decision Tree Decision Boundary on PCA-Reduced Data

B. Implications for Model Selection

Non-linear separability of classes means that non-complex models such as Logistic Regression may struggle to generalize high accuracy using this data set. Non-complex models capable of generalizing to non-linear relationships, such as Decision Trees, Random Forests, or ensemble learning based on algorithms such as AdaBoost are better equipped. Our experiments confirmed this by opting for Decision Trees and Random Forest over Logistic Regression both in terms of accuracy as well as the F1-score.

C. Limitations and Future Work

While our study provides valuable insights, there are several limitations:

- The dataset is relatively small, which may limit the generalizability of the results.
- The class imbalance in the target variable could affect model performance, particularly for the minority class (4 or more doctors).
- The use of PCA for visualization may oversimplify the underlying structure of the data, as it only captures linear relationships.

Future work could explore:

- Advanced techniques for handling class imbalance, such as SMOTE or class weighting.
- The use of deep learning models, such as neural networks, to capture more complex patterns in the data.
- Incorporating additional features or external datasets to improve model performance.

D. Final Remarks

Lastly, our research demonstrates the efficacy of non-linear machine learning models in forecasting the elderly population's healthcare utilization. Visualization of decision boundaries by PCA and Decision Trees illustrates the non-linear separability of classes, thus the need for sophisticated modeling techniques. Our results supplement the existing literature on data-driven health solutions and lay the foundation for further research in this area.

REFERENCES

[1] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[2] 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.

[3] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. New York: Springer, 2009.

[4] D. W. Hosmer and S. Lemeshow, *Applied Logistic Regression*, 2nd ed. New York: Wiley, 2000.

[5] 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.

[6] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[7] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[8] Z.-H. Zhou, "Ensemble methods: Foundations and algorithms," *Chapman and Hall/CRC*, 2012.

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[10] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.