Machine Learning

Course Project Report

(Phase-II)

Title of the project: National Poll on Healthy Aging

Student : Shagun Shukla, shagun.s-26@scds.saiuniversity.edu.in

ML Category:
Classification

---

1. Introduction

Donated on December 5, 2023, the National Poll on Healthy Aging (NPHA) dataset is a carefully selected subset created especially to test and build machine learning algorithms for estimating how many doctors a survey participant will visit in a given year. The replies from seniors who took part in the NPHA survey make up this dataset, which provides insightful information about their healthcare needs and behaviours.

The potential to improve healthcare planning and resource allocation makes modelling the relationship between the many characteristics of an older adult's profile and their healthcare utilisation important. This research can help healthcare practitioners and policymakers better understand the healthcare needs of the ageing population by identifying important factors that influence the number of doctor visits. This will ultimately improve the quality and accessibility of care.

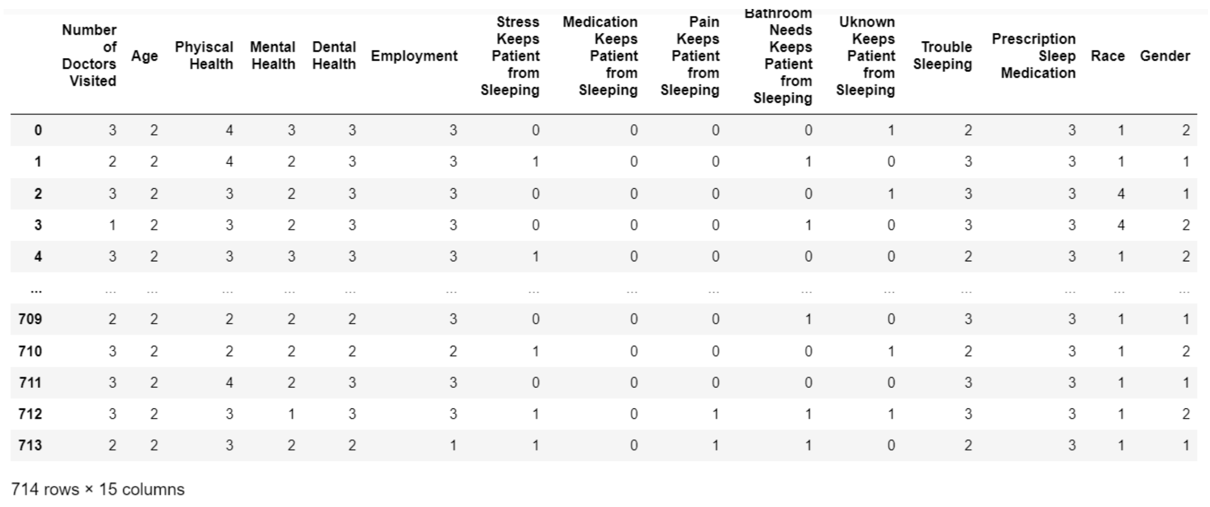
- **Problem Statement:** (Classification)
  Predicting the number of doctors a senior sees annually based on various features from their survey responses.

2. Dataset and Features

- Dataset Details: The National Poll on Healthy Aging (NPHA) dataset, which was given on December 5, 2023, is used in this research. This dataset is a subset that has been specially filtered in order to create and test machine learning algorithms that predict how many doctors a survey participant would visit in a given year. There are 14 columns and 714 rows in the dataset. The columns show the respondents' various characteristics, including age, employment position, physical, mental, and dental health, and sleep problems, while the rows show examples of senior survey responses.
  Thirteen of the fourteen columns are input features for our learning model; the goal variable that

we want to predict is the column labelled "Number of Doctors Visited." The number of various doctors a senior sees in a year is indicated by this target variable. The sample points of the target variable have been encoded in the range (0 to 2) using the *LabelEncoder()*.

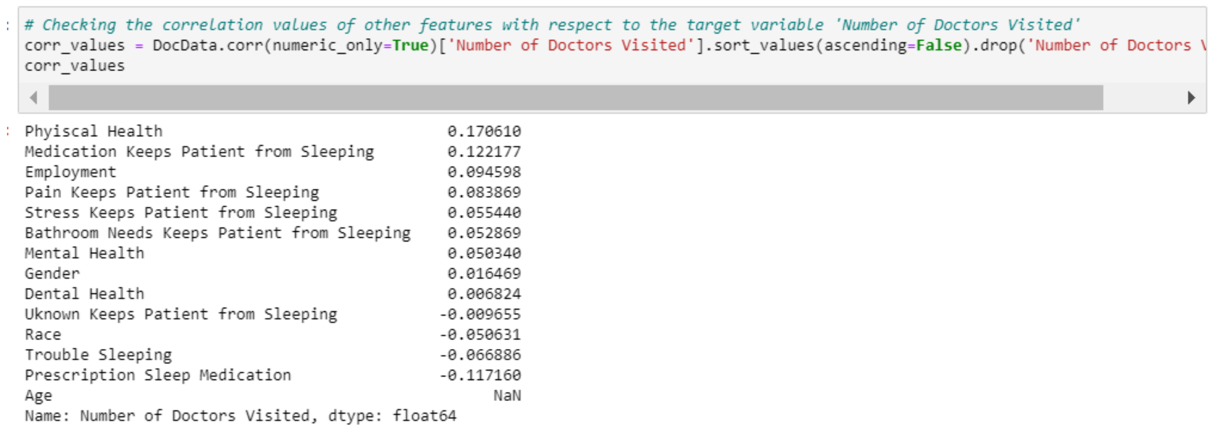
- Exploratory Data Analysis: Initially, the dataset contains 714 rows and 15 columns.



	Number of Doctors Visited	Age	Phyiscal Health	Mental Health	Dental Health	Employment	Stress Keeps Patient from Sleeping	Medication Keeps Patient from Sleeping	Pain Keeps Patient from Sleeping	Bathroom Needs Keeps Patient from Sleeping	Uknown Keeps Patient from Sleeping	Trouble Sleeping	Prescription Sleep Medication	Race	Gender
0	3	2	4	3	3	3	0	0	0	0	1	2	3	1	2
1	2	2	4	2	3	3	1	0	0	1	0	3	3	1	1
2	3	2	3	2	3	3	0	0	0	0	1	3	3	4	1
3	1	2	3	2	3	3	0	0	0	1	0	3	3	4	2
4	3	2	3	3	3	3	1	0	0	0	0	2	3	1	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
709	2	2	2	2	2	3	0	0	0	1	0	3	3	1	1
710	3	2	2	2	2	2	1	0	0	0	1	2	3	1	2
711	3	2	4	2	3	3	0	0	0	0	0	3	3	1	1
712	3	2	3	1	3	3	1	0	1	1	1	3	3	1	2
713	2	2	3	2	2	1	1	0	1	1	0	2	3	1	1

714 rows × 15 columns

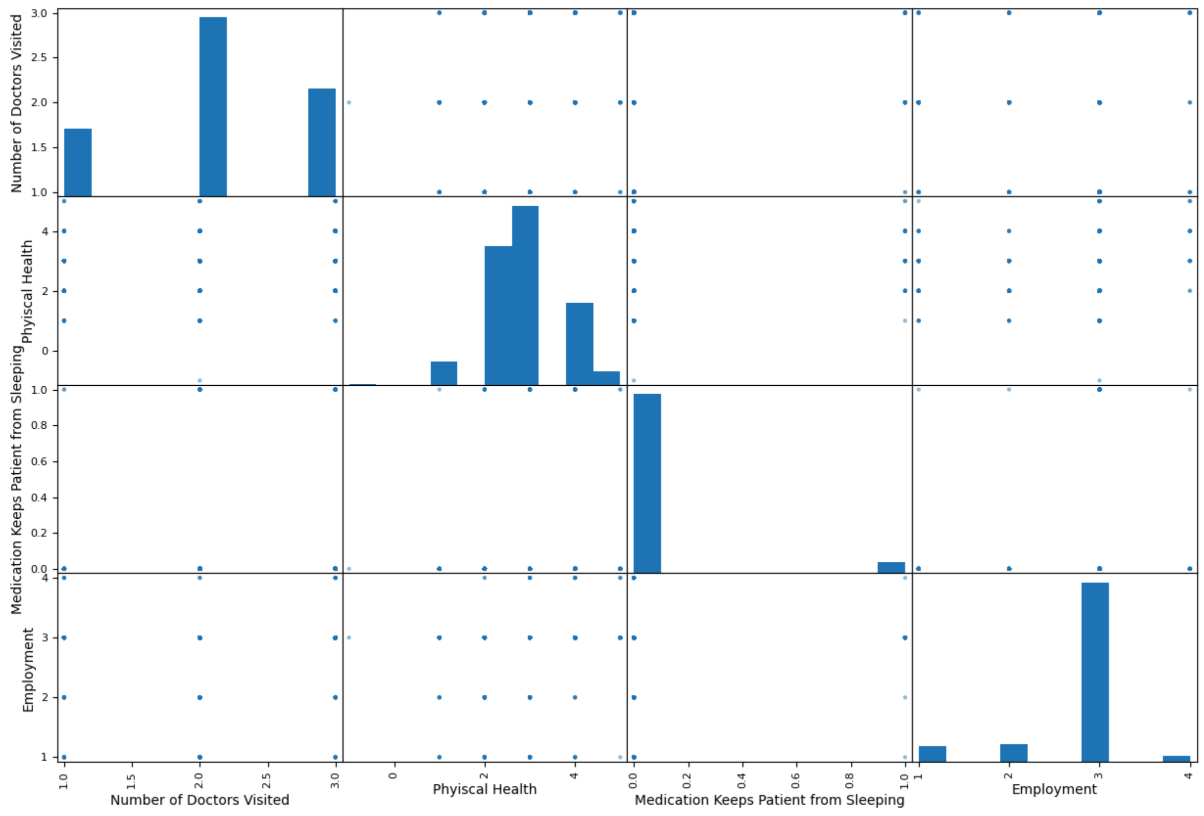
The target variable is one of the columns that has the data type Int64. The dataset has **42 duplicate rows**, which when eliminated, makes the size of the dataset as (672, 14). There are **no missing** values in the dataset. Number of Doctors Visited is the target variable. Its values range from 1 at the least to 2 at the median, 2.113095 at the mean, and 3 at the maximum. Examining the correlation values between the input features and the target variable, we find that all features have correlation values less than 0.1, with the exception of "Physical Health," which has the highest correlation value at 0.170610, and "Medication Keeps Patient from Sleeping," which has a correlation value of 0.122177. This suggests that the qualities have relatively little linear reliance on one another between the attributes and the target variable.



```
# Checking the correlation values of other features with respect to the target variable 'Number of Doctors Visited'
corr_values = DocData.corr(numeric_only=True)['Number of Doctors Visited'].sort_values(ascending=False).drop('Number of Doctors V
corr_values
```

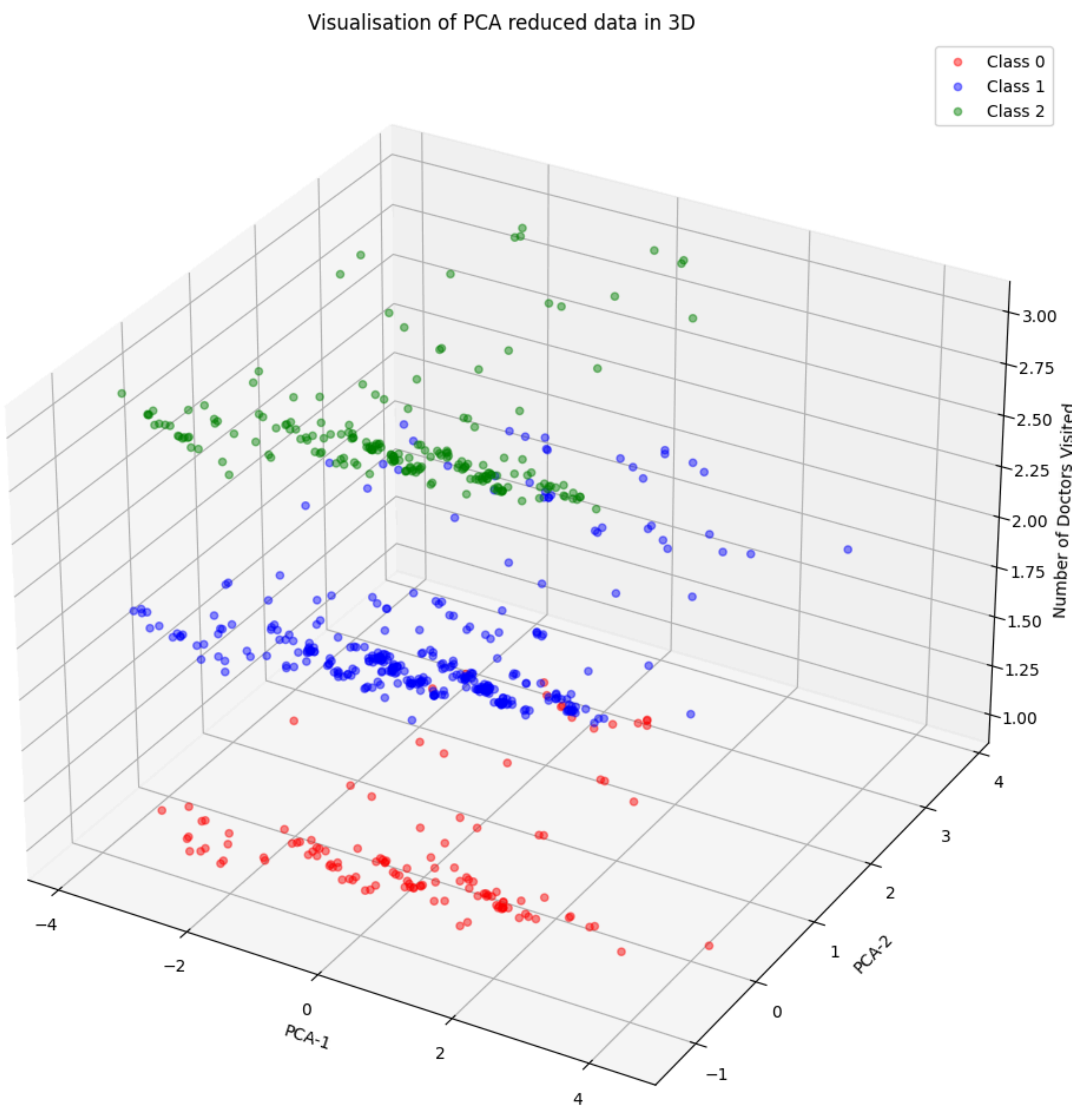
```
Phyiscal Health                              0.170610
Medication Keeps Patient from Sleeping       0.122177
Employment                                   0.094598
Pain Keeps Patient from Sleeping             0.083869
Stress Keeps Patient from Sleeping           0.055440
Bathroom Needs Keeps Patient from Sleeping   0.052869
Mental Health                                0.050340
Gender                                       0.016469
Dental Health                                0.006824
Uknown Keeps Patient from Sleeping          -0.009655
Race                                        -0.050631
Trouble Sleeping                            -0.066886
Prescription Sleep Medication               -0.117160
Age                                               NaN
Name: Number of Doctors Visited, dtype: float64
```

Plotting the scatter matrix of the top 3 correlated features and target variable 'Numbers of Doctors Visited'



- <u>Dimensionality Reduction & Data Visualisation:</u>
  We use dimensionality reduction techniques to reduce the number of features in the data from 14 to 3, so that the point cloud can be visualised in 3D when the features are plotted against the target variable 'Number of Doctors Visited'. In this project, we only implemented Principal Component Analysis (PCA). The below plot shows the distribution of the data points with respect to the PCA features, differentiated by colour based on their class - red implies 'Class 0', blue implies 'Class 1' and green implies 'Class 2', where 'Class 0' means 'Number of Doctors Visited' is 0-1, and 2-3 and 4

or more for ‘Class 1’ and ‘Class 2’ respectively.



3. Methods

Following are the various methods used in this project.

3.1 Baseline - Linear Regression or Logistic/Softmax Regression based on the ML category

Logistic regression extended for many classes is called softmax regression. Only binary categorization is supported by logistic regression. Regression using softmax involves an estimate of a score given for each class k. The probability of each class is then estimated using the softmax function. The class with the highest estimated probability is the one that is predicted.

I decided to make only three classes for our output data, since the accompanying information about the features stated three classes, and I wanted to maintain consistency with that. I performed Softmax regression on the full training dataset, standardised, and the PCA reduced (10) dataset. Softmax regression on the regular data gave us an accuracy of 0.57143, standardised data gave us 0.56548, and the experiment with PCA reduced data gave 0.55357.
The observation that Softmax regression performs best on regular as compared to the Standardised and PCA reduced one, which becomes quite clear here with respect to our specific dataset.

3.2 Support Vector Machines

● Linear SVM and Kernel SVM

The best hyperplane is determined using linear SVC by locating the support vectors that are closest to the decision boundary. It differs from the SVC class with linear kernels in that it employs a one-versus-rest classification scheme, whereas the latter employs a one-versus-one strategy. The technique seeks to minimise classification errors while optimising the margin—that is, the separation between the support vectors and the decision boundary. We only used the tolerance and the C parameters, or the hardness/softness parameter, despite the fact that it has other hyperparameters. After observing a very low correlation between the input and output data, we made the decision to allow for some misclassification in order to obtain a respectable score. For the Hyperparameter C, we have used C=2 as hyperparameter as default value for all the three types of data, however in case we don't provide any parameter value, it will take random values every time we run the code and will result in different Classification accuracies that are strongly correlated. For the regular, standardised, and PCA-reduced data of ten characteristics, we employed LinearSVC. We obtained accuracy ratings of 0.55357, 0.56548 and 0.54167, respectively, for Regular, Standardised and PCA reduced data while employing LinearSVC. In our situation, the regular and standardised data performed almost equally and better than PCA reduced data. Similarly, for Linear kernel we got an accuracy of 0.50595 for all three categories using linear kernel SVC, or the SVC class with kernel='linear'

● Polynomial and Radial Basis Function kernels

Using a polynomial function, the Polynomial Kernel SVC maps the original data into a higher-dimensional feature space. This enables the SVC to determine a linearly separable linear decision boundary in a higher dimension, which is comparable to a polynomial boundary in the original (lower) dimension. The polynomial function that was employed and the model both have the same degree. While lesser degrees may lead to underfitting, higher degrees can help identify more complex patterns in the data. As a result, considering the characteristics of our dataset, the polynomial degree selection should be carefully considered. The obtained accuracy was roughly 0.52976, 0.5119 and 0.50595 for regular, standardised and PCA reduced data respectively. Here also we used C=2 as default value for our hyperparameter and kernel='poly'.

A kernel function called the Radial Basis Function transfers the data onto higher dimensions once more. It does this by choosing random data points, drawing Gaussian distributions over them, and then classifying the data based on how similar the distributions are to each other. It returns to its initial dimensions after a suitable division is identified using the provided hyperparameters, but this time there is a distinct class division. For model training, we haven't provided any hyperparameter value with gamma='scale' , without making any adjustments. Achieved accuracy was 0.51786,0.50595 and 0.52976 for regular, standardised and PCA reduced data respectively and kernel='rgf'. Here also we used C=2 and gamma ='scale' as default values for our hyperparameter.

3.3 Decision Tree

A non-parametric machine learning method for both regression and classification issues is decision trees. The term "non-parametric" indicates that the number of parameters varies. In terms of computing intensity and prediction performance, Decision Trees are thought to be superior to the earlier methods employed in this project. They operate by analysing the data's properties and creating a binary tree-based decision model that is used to generate predictions. The *CART Algorithm* is the one used to train decision trees.
A greedy technique called the Classification and Regression Tree (CART) technique is used to train Decision Trees. To minimise the *Gini impurity* of the split, it operates by recursively dividing the training data into two subsets based on a feature and corresponding threshold.
The ratio of class instances among all training examples within a specific node is captured by the *Gini* value, which quantifies the impurity within a node. It is computed with the following formula:

$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$

The accuracy obtained was 0.51786, 0.51786, and 0.48214 for regular, standardised and PCA reduced Data respectively. As parameters we have made criterion='gini' and random state =42 , also for our hyperparameter, 'max_leaf_nodes' = 20.

3.4 Random Forest

A forest, or collection of Decision Trees, is subjected to Random Forest classification, which employs the Ensemble Learning technique of Bagging. The Random Forest classification technique enhances prediction accuracy and reduces overfitting by training many Classification Trees and selecting the mode value of their predictions. Because Random Forest models leverage the strength of Ensemble Learning on the high performance of Decision Trees, they are regarded as some of the greatest Machine Learning techniques available. As parameters we have made n_jobs=-1 and random state =42 , also for our hyperparameter, 'max_leaf_nodes' = 20 and 'n_estimators'=100.

For the normal, standardised, and PCA reduced data, the corresponding scores were 0.5297619047619048, 0.5297619047619048, and 0.5416666666666666, respectively.

3.5 AdaBoost

Another well-liked ensemble learning method is called "boosting," which creates a strong learner by combining many "weak" learners (predictors). It is regarded as the most effective machine learning method. In contrast to Bagging, which involves running many predictors individually and averaging their results, Boosting trains predictors one after the other so that each one makes up for the flaws of the preceding one. Thus, even when every learner performs poorly, the model as a whole produces remarkably accurate predictions.
Adaptive Boosting, or AdaBoost, is a technique where the model sequentially adjusts to each poor learner's performance. It accomplishes this by altering each subsequent learner to fix the mistakes made by the one before it. This is accomplished by allocating weights based on the size of the prediction mistake.
The score obtained was 0.41666666666666667, 0.4226190476190476, and 0.5 for regular, standardised and PCA reduced Data respectively. As parameters we have made learning_rate=0.5, algorithm='SAMME.R' and random state =42 , also for our hyperparameter, 'max_leaf_nodes' = 20 for DecisionTreeClassifier we have 'n_estimators'=100.

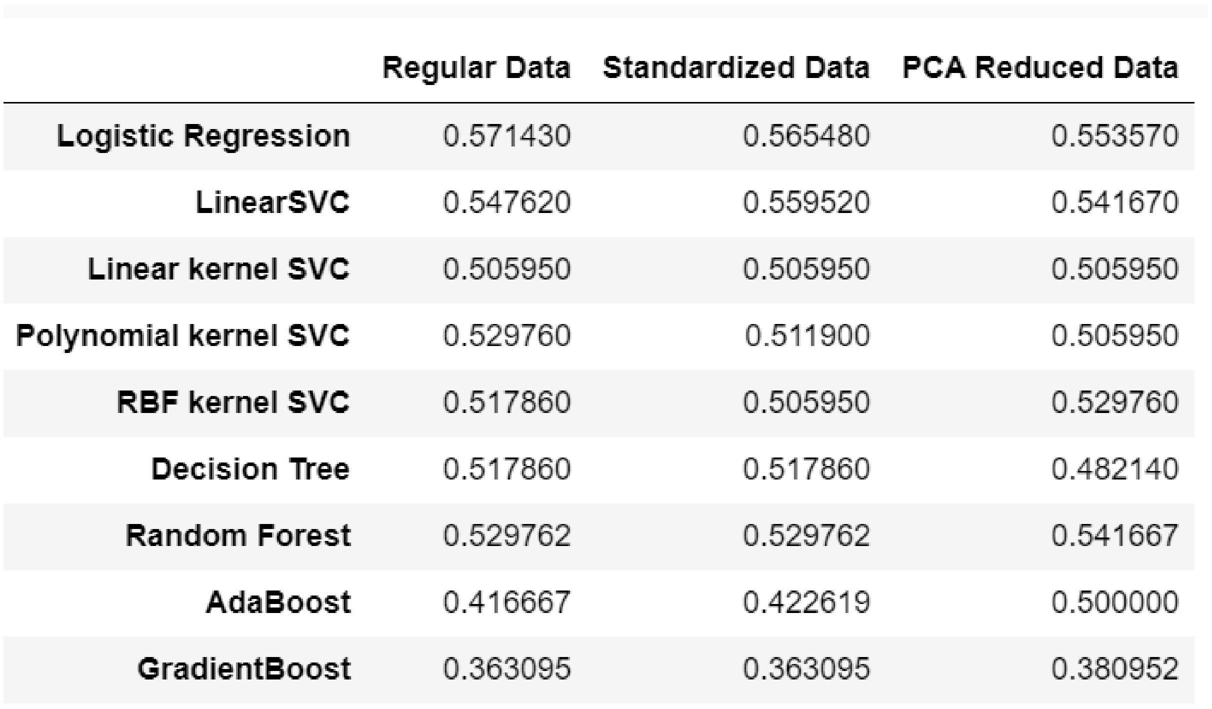
3.6 Gradient Boosting

A more versatile version of AdaBoost known as gradient boosting is capable of using any loss function in place of AdaBoost's proprietary exponential one. It becomes more resilient to outliers and versatile as a result. Gradient Boosting specifically uses the residual error of the preceding learner to train each successive weak learner. It uses gradients to identify vulnerabilities rather than assigning weights to data points.
The obtained R2 scores for the regular, standardised, and PCA reduced data were 0.3630952380952381, 0.3630952380952381, and 0.38095238095238093, respectively. As parameters we have made learning_rate=0.5, and random state =42 , also for our hyperparameter, 'max_leaf_nodes' = 20 and 'n_estimators'=100.

4. Results

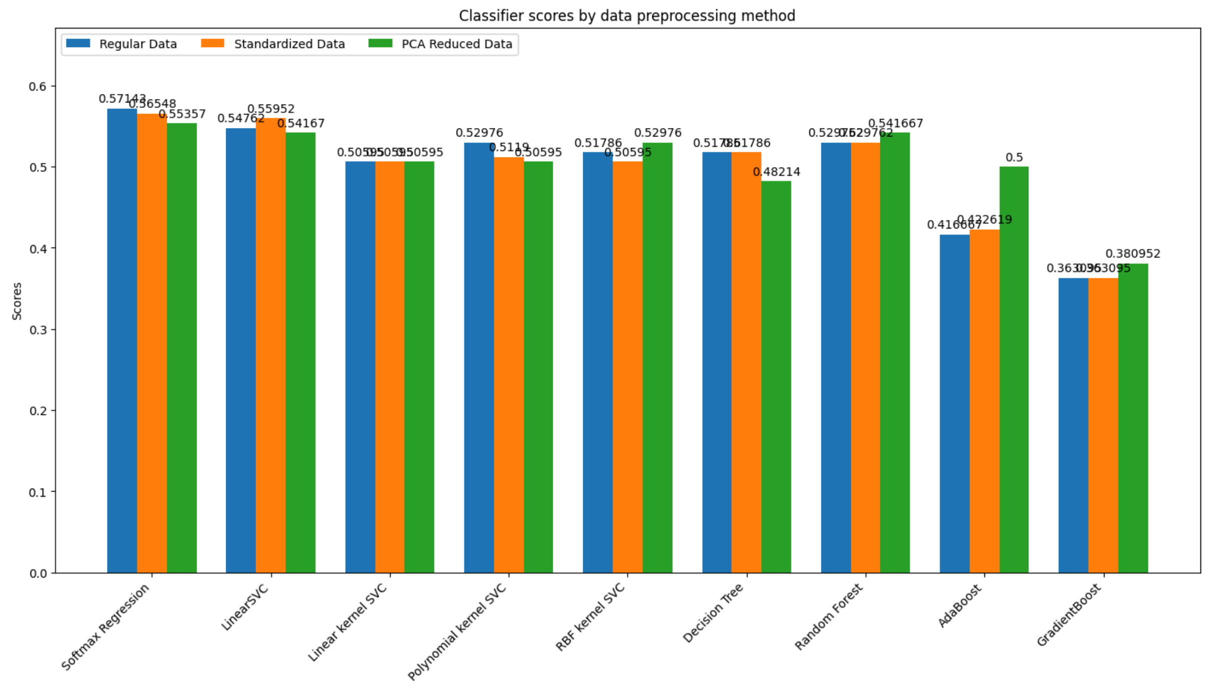
Classification Accuracy

Table of Classification Accuracy scores for all the Machine Learning experiments conducted in this project



	Regular Data	Standardized Data	PCA Reduced Data
Logistic Regression	0.571430	0.565480	0.553570
LinearSVC	0.547620	0.559520	0.541670
Linear kernel SVC	0.505950	0.505950	0.505950
Polynomial kernel SVC	0.529760	0.511900	0.505950
RBF kernel SVC	0.517860	0.505950	0.529760
Decision Tree	0.517860	0.517860	0.482140
Random Forest	0.529762	0.529762	0.541667
AdaBoost	0.416667	0.422619	0.500000
GradientBoost	0.363095	0.363095	0.380952

<u>Bar Plot to display the above results:</u>

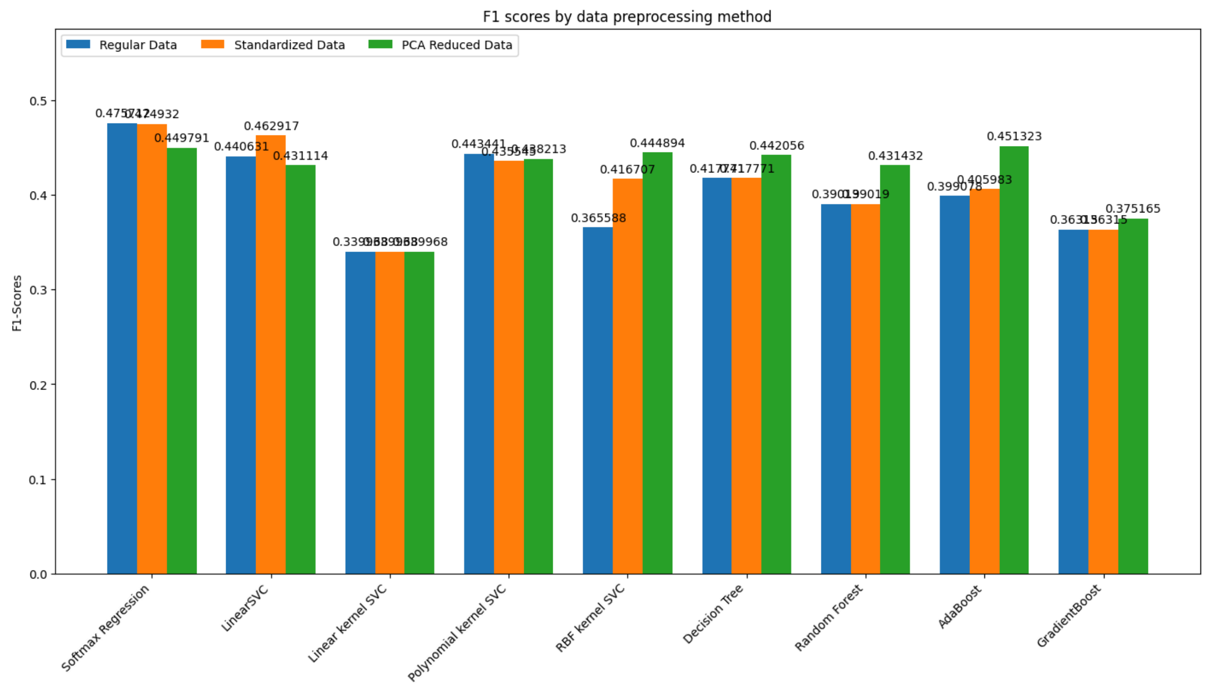


<u>F1- Score</u>

Table of F1 Scores for all the Machine Learning experiments conducted in this project

	Regular Data	Standardized Data	PCA Reduced Data
Logistic Regression	0.475712	0.474932	0.449791
LinearSVC	0.440631	0.462917	0.431114
Linear kernel SVC	0.339968	0.339968	0.339968
Polynomial kernel SVC	0.443441	0.435545	0.438213
RBF kernel SVC	0.365588	0.416707	0.444894
Decision Tree	0.417771	0.417771	0.442056
Random Forest	0.390190	0.390190	0.431432
AdaBoost	0.399078	0.405983	0.451323
GradientBoost	0.363150	0.363150	0.375165

<u>Bar Plot to display the above results:</u>



5. Hyperparameter Tuning

Hyperparameters are important in machine learning because they affect how well an estimator (such a classifier or regressor) performs. In contrast to model parameters, which are determined by training data, hyperparameters are externally defined and affect the behaviour of the model. These hyperparameters are supplied as arguments to scikit-learn when building an estimator.

Grid Search:

We specify a grid of hyperparameters and the ranges of values that correspond to them in Grid Search.
The method thoroughly assesses every conceivable pairing of these hyperparameter values.
Finding the best combination to maximise a performance metric (such the cross-validated F1 score) is the aim.
Grid Search's thorough search can make it computationally costly.

Randomised Search:

One variant of grid search is randomised search.
It selects hyperparameter values at random from the given grid, as opposed to assessing every possible combination.
This explores a wide range of hyperparameter settings while minimising compute overhead.
When conducting a thorough search is not feasible, the use of Randomised Search comes in handy.

The experiment employed both tuning procedures on three distinct dataset types: Standardised, PCA Reduced, and Features Selected Data.

5.1 SVM with RBF kernel

RBF Kernel

Explanation of the Hyperparameters Tuned

C: The regularisation parameter (inverse of the regularisation strength). It controls the trade-off between achieving a low training error and a low testing error. Higher values of C lead to less regularisation.
gamma: Kernel coefficient for 'rbf', 'poly', and 'sigmoid'. It defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’.
We will explore different values of `C` and `gamma` to find the optimal settings.

- **Grid Search CV:** The grid search is performed using a range of values for `C` and `gamma`. Specifically, we create a parameter grid with `C` ranging from 1 to 9 and `gamma` ranging from 0.1 to 1.0 in 10 evenly spaced intervals. The optimal values for the hyperparameters `C` and `gamma` are found to be 1.0 and 0.1, respectively. The cross-validated classification accuracy achieved by the best configuration is approximately **0.523810.**

5.2 Decision Trees

Explanation of the Hyperparameters Tuned

- **max_depth**: The maximum depth of the tree. This parameter controls how deep the tree can grow. Deeper trees can capture more complex patterns but are more likely to overfit.
- **max_leaf_nodes**: The maximum number of leaf nodes in the tree. Limiting the number of leaf nodes can prevent overfitting.
- **random_state**: Controls the randomness of the estimator. Setting a fixed value for reproducibility.

We will explore different values of `max_depth` and `max_leaf_nodes` to find the optimal settings.

Grid Search CV: The grid search is performed using a range of values for `max_depth` and `max_leaf_nodes`. Specifically, we create a parameter grid with `max_depth` ranging from 3 to 13 in steps of 2, `max_leaf_nodes` ranging from 1 to 8, and `random_state` fixed at 42. The optimal values for the hyperparameters `max_depth` and `max_leaf_nodes` are found to be 3 and 8, respectively. The cross-validated classification accuracy achieved by the best configuration is approximately **0.517857.**

5.3 Random Forest

Explanation of the Hyperparameters Tuned

- **max_leaf_nodes**: The maximum number of leaf nodes in the tree. Limiting the number of leaf nodes can help control the complexity of the tree and prevent overfitting.
- **n_estimators**: The number of trees in the forest. More trees can improve the model's performance but also increase computational cost.
- **n_jobs**: The number of jobs to run in parallel. Setting this to -1 uses all available processors.
- **random_state**: Controls the randomness of the estimator. Setting a fixed value ensures reproducibility.

We will explore different values of max_leaf_nodes and n_estimators to find the optimal settings.

Grid Search CV: The grid search is performed using a range of values for max_leaf_nodes and n_estimators. Specifically, we create a parameter grid with max_leaf_nodes ranging from 32 to 64, n_estimators ranging from 100 to 200, n_jobs fixed at -1, and random_state fixed at 42. The optimal values for the hyperparameters max_leaf_nodes and n_estimators are found to be 32 and 200, respectively. The cross-validated classification accuracy achieved by the best configuration is approximately **0.523810.**

5.4 AdaBoost

Explanation of the Hyperparameters Tuned

- **learning_rate**: This parameter controls the contribution of each classifier. Lower values reduce the impact of each individual classifier, while higher values increase it.
- **n_estimators**: The number of boosting stages to be run. More stages can improve performance but also increase the risk of overfitting.
- **estimator__max_leaf_nodes**: The maximum number of leaf nodes for the base estimator, which is a decision tree in this case. Limiting the number of leaf nodes can help control the complexity of the base estimator and prevent overfitting.
- **algorithm**: Specifies the boosting algorithm to use. 'SAMME.R' uses a real boosting algorithm.
- **random_state**: Controls the randomness of the estimator. Setting a fixed value ensures reproducibility.

We will explore different values of `learning_rate`, `n_estimators`, and `estimator__max_leaf_nodes` to find the optimal settings.

Grid Search CV: The grid search is performed using a range of values for `learning_rate`, `n_estimators`, and `estimator__max_leaf_nodes`. Specifically, we create a parameter grid with `learning_rate` set to `[0.01, 0.05, 0.1, 0.5, 1]`, `n_estimators` set to [50, 100, 150, 200], `estimator__max_leaf_nodes` set to 32, `algorithm` set to 'SAMME.R', and `random_state` fixed at 42. The optimal values for the hyperparameters `learning_rate` and `n_estimators` are found to be 0.01 and 100, respectively. The cross-validated classification accuracy achieved by the best configuration is approximately **0.505942.**

5.5 Gradient Boosting

Explanation of the Hyperparameters Tuned

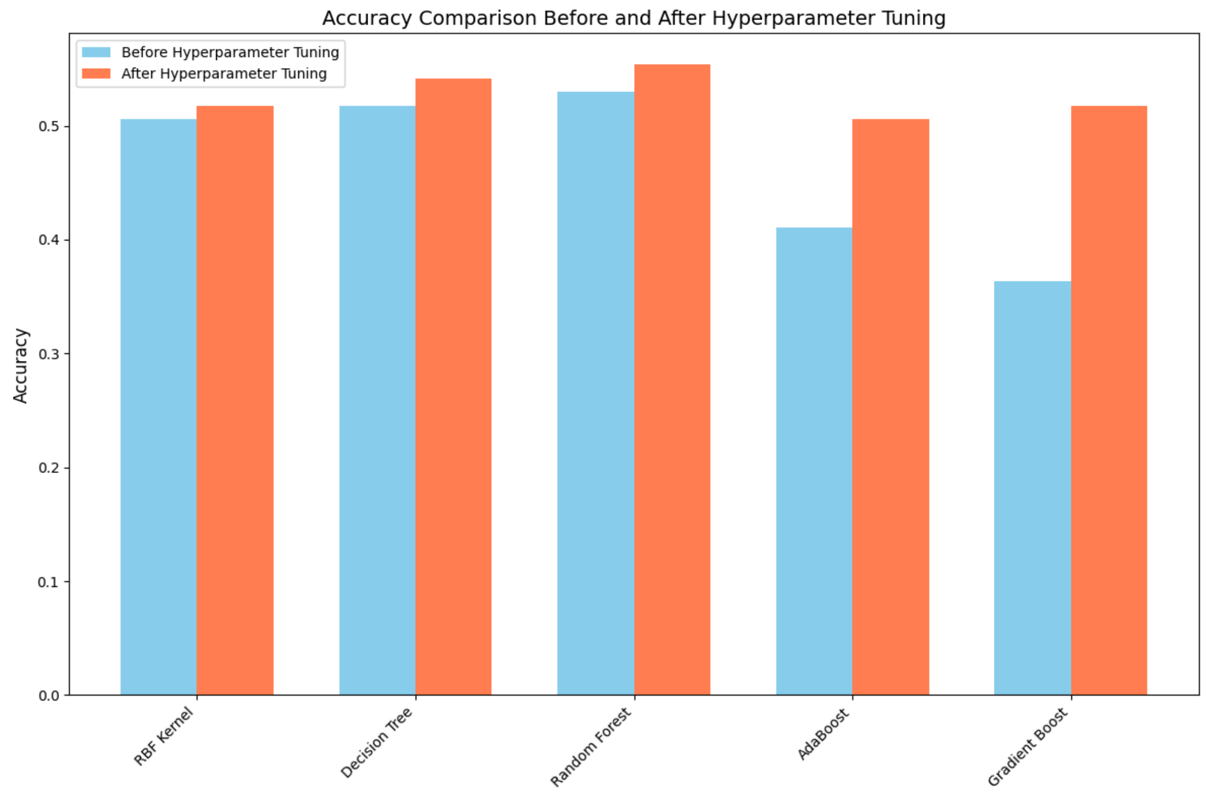
- **learning_rate**: This parameter controls the contribution of each tree. Lower values reduce the impact of each individual tree, while higher values increase it.
- **n_estimators**: The number of boosting stages to be run. More stages can improve performance but also increase the risk of overfitting.
- **max_leaf_nodes**: The maximum number of leaf nodes for each base estimator (a decision tree in this case). Limiting the number of leaf nodes can help control the complexity of the base estimator and prevent overfitting.
- **random_state**: Controls the randomness of the estimator. Setting a fixed value ensures reproducibility.

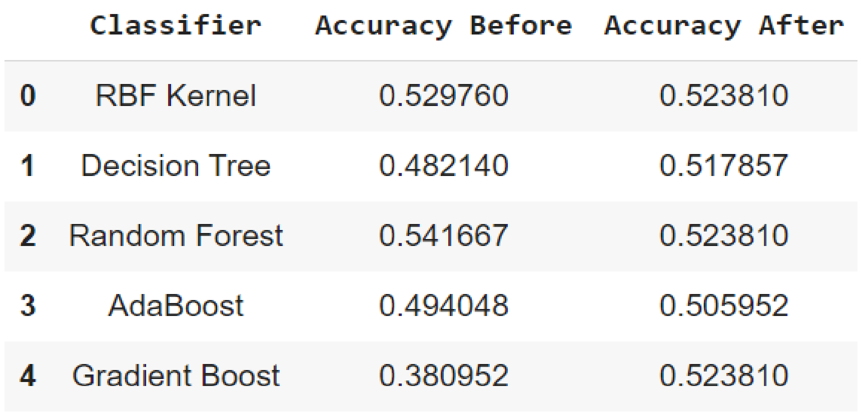
We will explore different values of learning_rate, n_estimators, and max_leaf_nodes to find the optimal settings.

Grid Search CV: The grid search is performed using a range of values for learning_rate, n_estimators, and max_leaf_nodes. Specifically, we create a parameter grid with learning_rate set to [0.01, 0.05, 0.1], n_estimators set to [100, 150, 200], max_leaf_nodes set to 32, and random_state fixed at 42. The optimal values for the hyperparameters learning_rate and n_estimators are found to be 0.01 and 100, respectively, with max_depth as 3. The cross-validated classification accuracy achieved by the best configuration is approximately **0.523810.**

6. Results after hyperparameter tuning

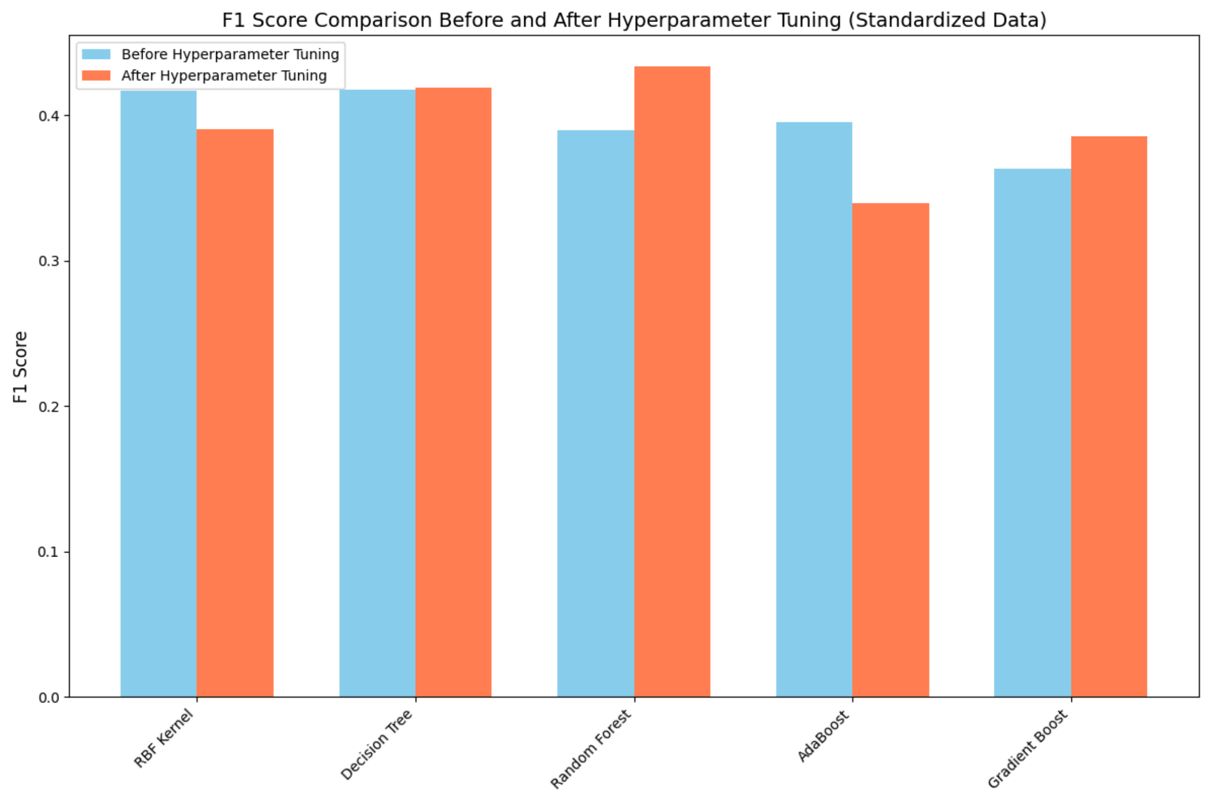
Classification Accuracy:





	Classifier	Accuracy Before	Accuracy After
0	RBF Kernel	0.529760	0.523810
1	Decision Tree	0.482140	0.517857
2	Random Forest	0.541667	0.523810
3	AdaBoost	0.494048	0.505952
4	Gradient Boost	0.380952	0.523810

<u>F1 - Scores</u>



	Classifier	F1 Before	F1 After
0	RBF Kernel	0.416707	0.390662
1	Decision Tree	0.417771	0.419194
2	Random Forest	0.390190	0.433460
3	AdaBoost	0.395717	0.339968
4	Gradient Boost	0.363150	0.385415

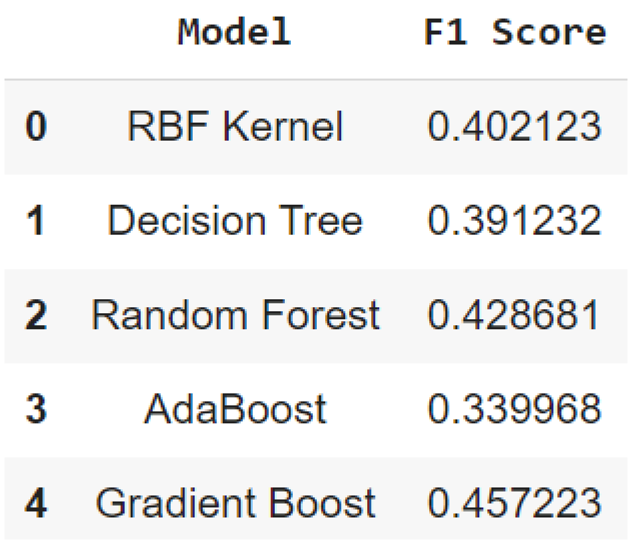
7. Feature Reduction

To further improve the F1-score, we investigated dimensionality reduction using Principal Component Analysis (PCA). With 14 features, PCA was applied to the training data, retaining the top 7 principal components (50% of original variance) to create a more efficient feature set. The best performing SVC model from GridSearchCV was then re-trained on this reduced data, and its F1-score on the test set will be compared to the results without PCA in the next section. Based on the F1 score scored by the models in the previous section, **Random Forest** performed the best, even on the PCA reduced data it outperformed most of the models as shown below with a F1 score of **0.523810.**

Classification Accuracy after Hyperparameter Tuning (PCA Reduced Data):

	Model	Accuracy
0	RBF Kernel	0.523810
1	Decision Tree	0.517857
2	Random Forest	0.523810
3	AdaBoost	0.505952
4	Gradient Boost	0.523810

F1 Score after Hyperparameter Tuning (PCA Reduced Data):



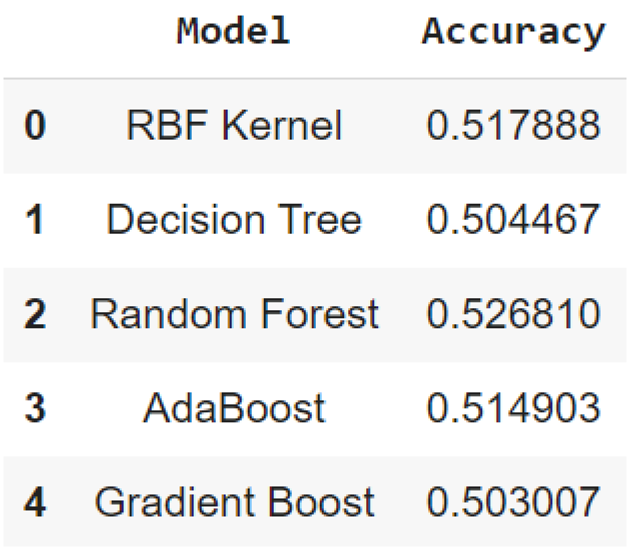
	Model	F1 Score
0	RBF Kernel	0.402123
1	Decision Tree	0.391232
2	Random Forest	0.428681
3	AdaBoost	0.339968
4	Gradient Boost	0.457223

8. Feature Selection

We further explored feature selection using the SelectPercentile method from scikit-learn. Since we have more than two features (7 in this case, after applying PCA), this approach was employed to retain the 50% of features with the highest scores based on a scoring function (often chi-squared for classification tasks). This aims to identify the most informative features that contribute most to the target variable.

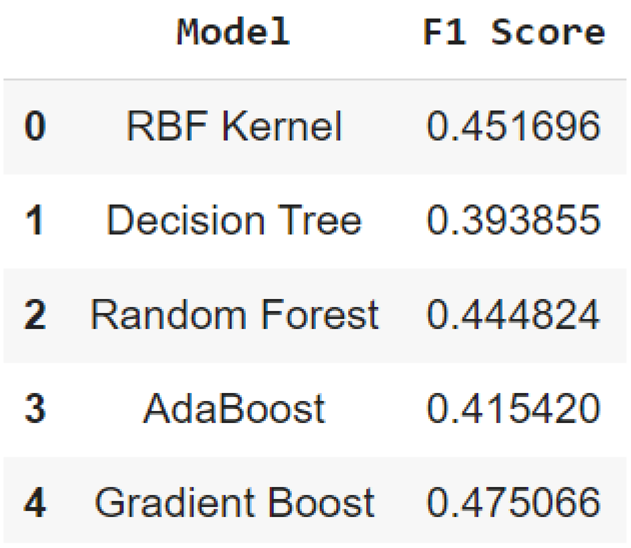
The models were then trained using the selected features: 'Physical Health', 'Employment', 'Stress Keeps Patient from Sleeping', 'Medication Keeps Patient from Sleeping', 'Pain Keeps Patient from Sleeping', 'Trouble Sleeping', and 'Prescription Sleep Medication'. Based on the F1 score scored by the models in the section 5, **Random Forest** performed the best, even on the Featured data it outperformed most of the models as shown below with a F1 score of **0.444824.**

Classification Accuracy after Hyperparameter Tuning (Feature Selected Data):



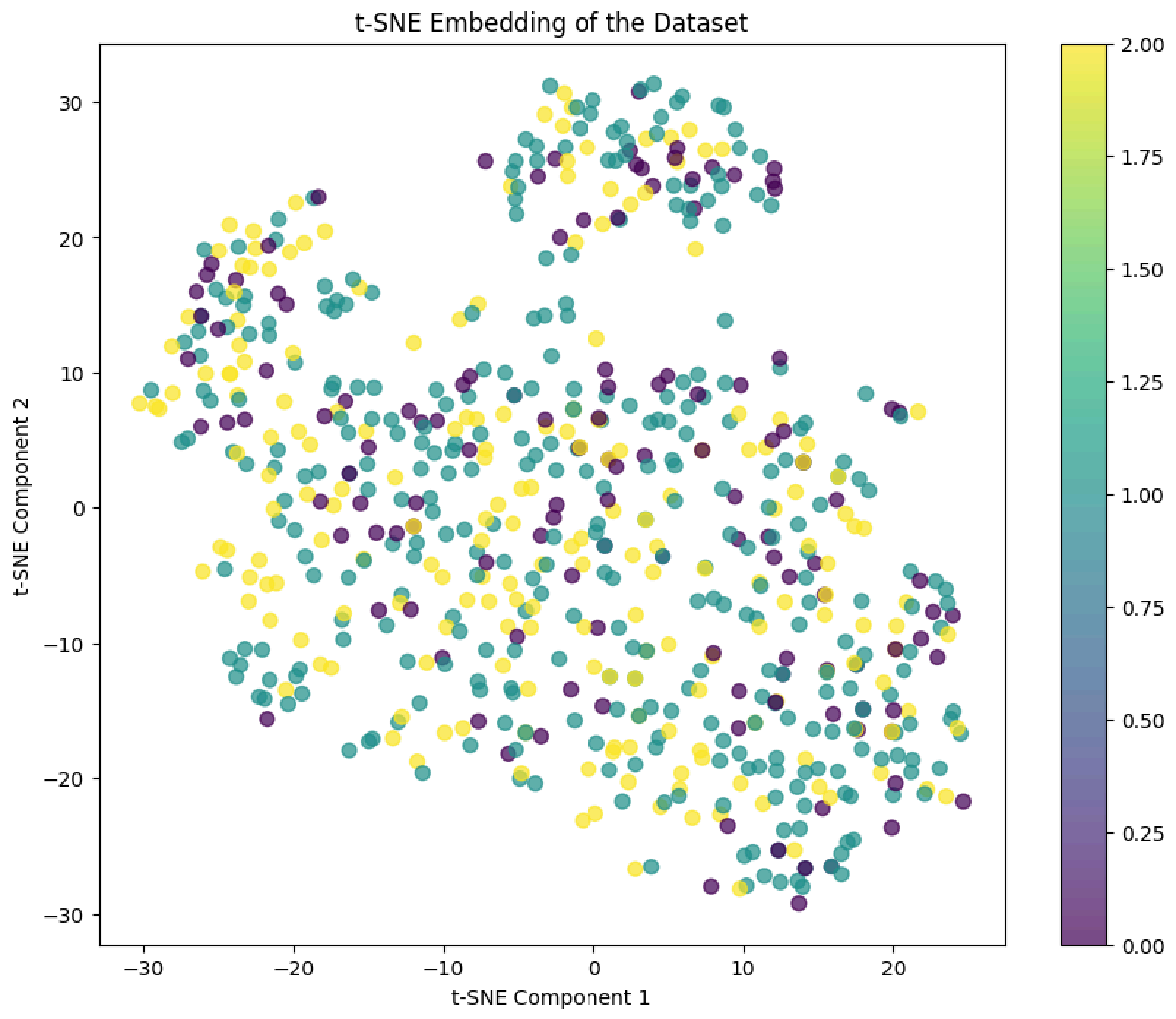
	Model	Accuracy
0	RBF Kernel	0.517888
1	Decision Tree	0.504467
2	Random Forest	0.526810
3	AdaBoost	0.514903
4	Gradient Boost	0.503007

F1 Score after Hyperparameter Tuning (Feature Selected Data):



	Model	F1 Score
0	RBF Kernel	0.451696
1	Decision Tree	0.393855
2	Random Forest	0.444824
3	AdaBoost	0.415420
4	Gradient Boost	0.475066

9. Data Visualization



10. Conclusion

Our experiments explored several machine learning algorithms for sleep problem classification and the impact of hyperparameter tuning and dimensionality reduction techniques. Hyperparameter tuning with GridSearchCV significantly improved all models. Principal Component Analysis (PCA) effectively reduced dimensionality while retaining performance for the best model (Random Forest). Feature selection with SelectPercentile resulted in optimum performance compared to using all features or PCA-reduced features. Random Forest achieved the strongest overall performance (accuracy: 0.523810, F1-score: 0.433460 on standardised data). Random Forest's success can be attributed to several factors: ensemble learning for increased robustness, handling high dimensionality effectively, and being less prone to overfitting. This project highlights the importance of hyperparameter tuning and dimensionality reduction in machine learning.

References:

[1] When and why to standardise a variable

[2] Scikit-Learn - Decision Tree Classifier

[3] Scikit-Learn - RandomForestClassifier

[4] Max_depth or Max_leaf_nodes in Random Forest?

[5] Adaboost vs Gradient Boosting - Data Science Stack Exchange