Machine Learning

Course Project Report

Title of the project: Parkinson's Disease Progression's Prediction (Using Machine Learning)

Student: Abhay Singh (abhay.s-26@scds.saiuniversity.edu.in)

ML Category: Regression

1. Introduction

Parkinson's disease is a chronic, progressive neurological disorder characterized by the degeneration of dopaminergic neurons in the brain, leading to motor symptoms such as tremors, rigidity, bradykinesia, and postural instability. Non-motor symptoms such as cognitive impairment, mood disorders, and autonomic dysfunction are also prevalent in patients. The Unified Parkinson's Disease Rating Scale (UPDRS) is widely used in clinical practice to assess the severity of Parkinson's disease symptoms. The total UPDRS score is a comprehensive measure that combines motor and non-motor aspects of the disease, providing a holistic view of disease progression.

The goal of this project is to predict the total UPDRS score using various regression models. Accurate prediction of this score can help in monitoring disease progression, adjusting treatment plans, and improving patient outcomes. Machine learning techniques offer a powerful tool for such predictions by identifying complex patterns in data that may not be apparent through traditional statistical methods. This project leverages a dataset containing various biomedical voice measurements, along with demographic and clinical features, to build regression models that can predict the total UPDRS score.

In Phase-I of this project, several regression models, including Linear Regression, Support Vector Regression (SVR) with different kernels, Decision Tree, Random Forest, AdaBoost, and Gradient Boosting, are evaluated using default settings. The dataset is split into training and testing sets to validate the performance of each model. The results are then analysed to determine which models perform best in predicting the total UPDRS score.

2. Dataset and Features

The dataset used in this project contains 5875 samples and 22 features. These features encompass a range of demographic, clinical, and biomedical voice measurements that are relevant to Parkinson's disease. The features include:

1. **subject#**: Unique identifier for each subject.

2. **age**: Age of the subject.
3. **sex**: Gender of the subject.
4. **test time**: Time since baseline (first recording).
5. **Motor UPDRS**: Motor sub score of the UPDRS.
6. **total_ UPDRS**: Total UPDRS score (target variable).
7. **Jitter (%), Jitter (Abs), Jitter**
   : Measures of variations in voice frequency.
   Shimmer, Shimmer(dB), Shimmer
   : Measures of variations in voice amplitude.
8. **NHR**: Noise-to-Harmonics ratio.
9. **HNR**: Harmonics-to-Noise ratio.
10. **RPDE**: Recurrence period density entropy.
11. **DFA**: Detrended fluctuation analysis.
12. **PPE**: Pitch period entropy.

The dataset is split into 75% training and 25% testing sets, and relevant feature scaling is applied to standardize the feature values.

3. Method:

Following are the various methods used in this project:

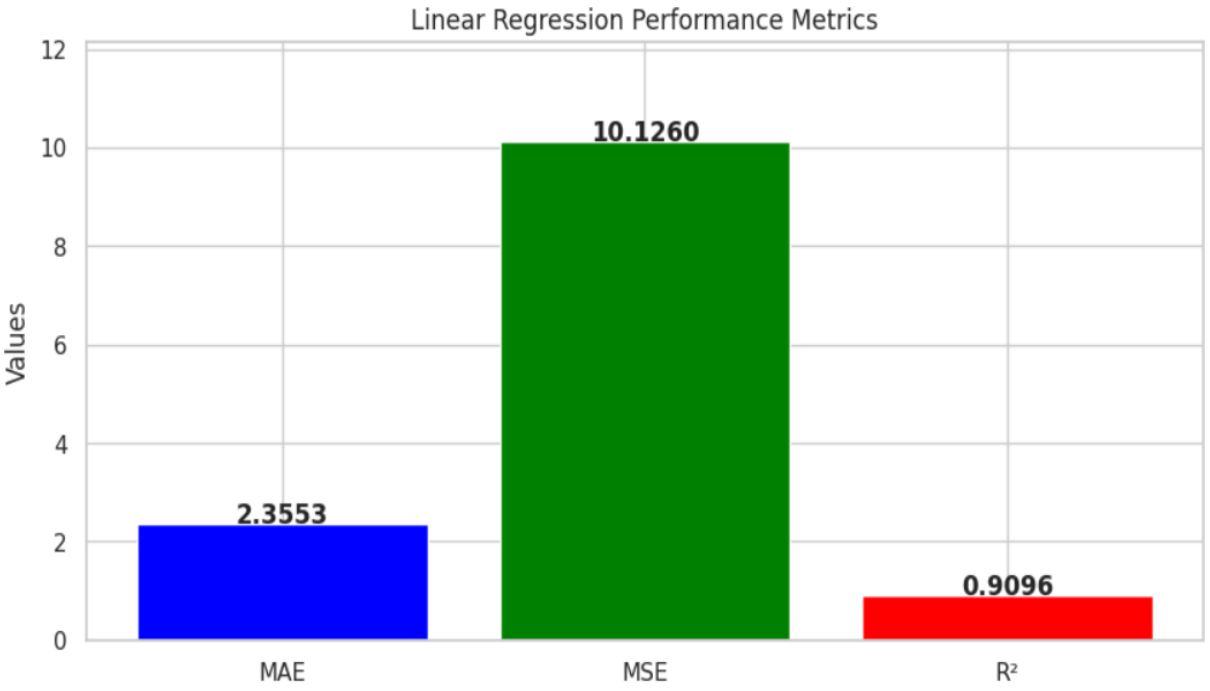
3.1 Baseline - Linear Regression

Linear regression is a fundamental and widely used method in machine learning and statistics for modelling the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the input features and the target variable. The goal of linear regression is to find the best-fitting linear equation that predicts the target variable as a weighted sum of the input features.

The linear regression model was applied to the dataset to predict the total UPDRS score. The dataset was split into 75% training and 25% testing sets, and feature scaling was performed to

standardize the feature values. The results obtained from the linear regression model are summarized below:

- **Mean Absolute Error (MAE) on Testing Set:** 2.3553
- **Mean Squared Error (MSE) on Testing Set:** 10.1260
- **R-squared (R^2) on Testing Set:** 0.9096



3.2 Support Vector Machines (SVM)

Brief description of the method:

Support Vector Machines (SVM) are a powerful set of supervised learning algorithms used for both classification and regression tasks. The fundamental idea behind SVM is to find a hyperplane that best separates the data into different classes (for classification) or predicts a continuous target variable (for regression). In the context of regression, the technique is known as Support Vector Regression (SVR).

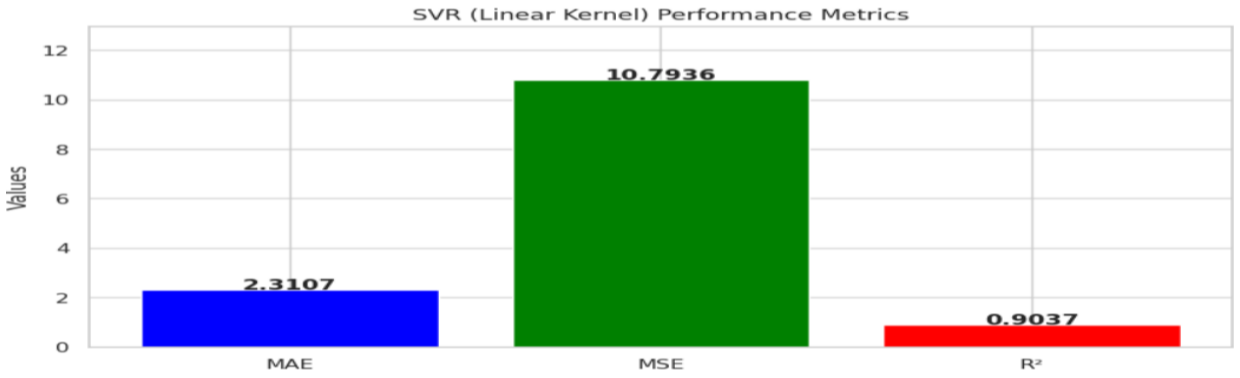
SVR attempts to find a function that approximates the target values within a specified margin of tolerance (epsilon). The algorithm tries to fit the best possible line (or hyperplane in higher dimensions) within this margin, while also ensuring that deviations outside the margin are minimized.

The SVR model can utilize different types of kernels to handle various types of data distributions. The three most commonly used kernels are:

1. Linear Kernel:

The linear kernel is the simplest type of kernel used in SVM. It is particularly useful when the data is linearly separable. In SVR with a linear kernel, the model attempts to find a linear hyperplane that best fits the data within the specified margin. The mathematical representation of the linear kernel is:

- **SVR with Linear Kernel:**
  - **Mean Absolute Error (MAE) on Testing Set:** 2.3107
  - **Mean Squared Error (MSE) on Testing Set:** 10.7936
  - **R-squared (R^2) on Testing Set:** 0.9037



1. Polynomial Kernel:

The polynomial kernel allows the SVM to fit nonlinear data by transforming the original feature space into a higher-dimensional space. This enables the model to capture complex relationships between the features and the target variable.

- **SVR with Polynomial Kernel:**
  - **Mean Absolute Error (MAE) on Testing Set:** 4.0567
  - **Mean Squared Error (MSE) on Testing Set:** 64.4453
  - **R-squared (R^2) on Testing Set:** 0.4248

2. Radial Basis Function (RBF) Kernel:

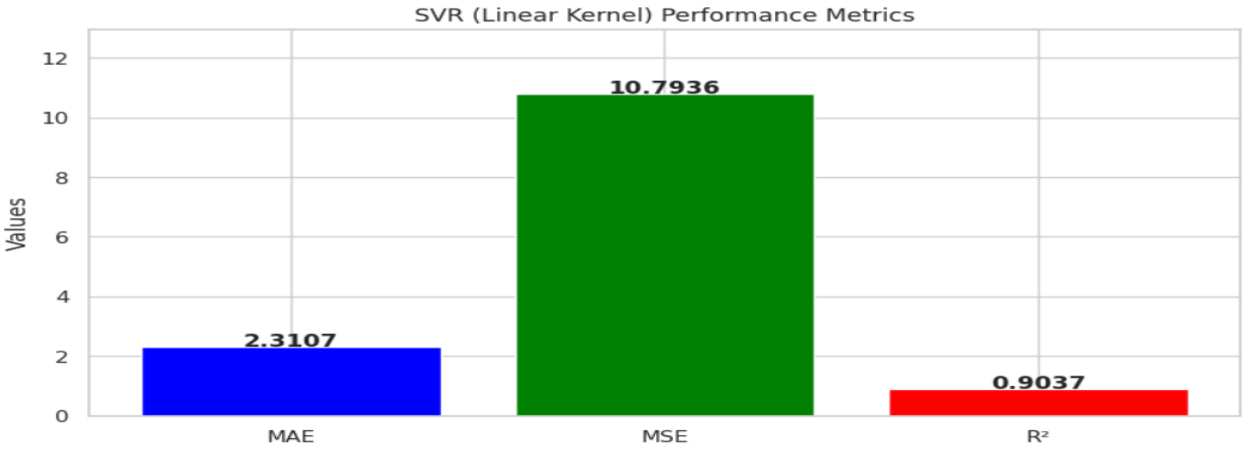
The RBF kernel, also known as the Gaussian kernel, is one of the most popular kernels used in SVM. It can handle nonlinear relationships by mapping the input features into an infinite-dimensional space. The RBF kernel is effective for capturing intricate patterns in the data.

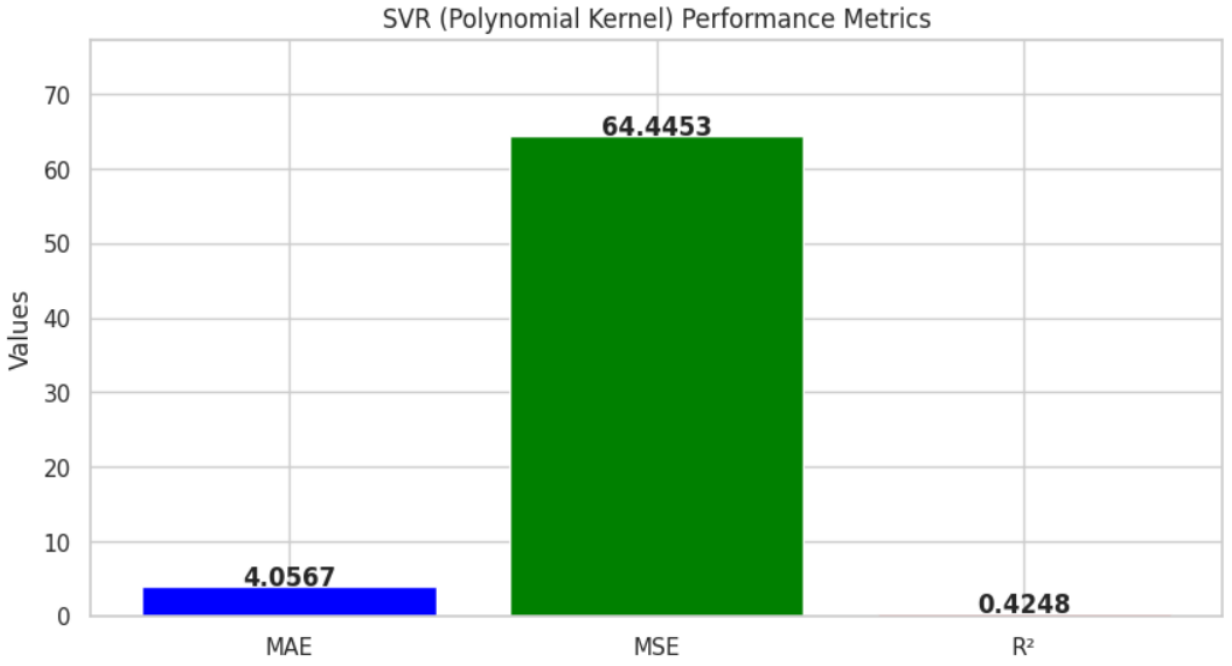
- **SVR with RBF Kernel:**
  - **Mean Absolute Error (MAE) on Testing Set:** 1.7998
  - **Mean Squared Error (MSE) on Testing Set:** 7.5082
  - **R-squared (R^2) on Testing Set:** 0.9330

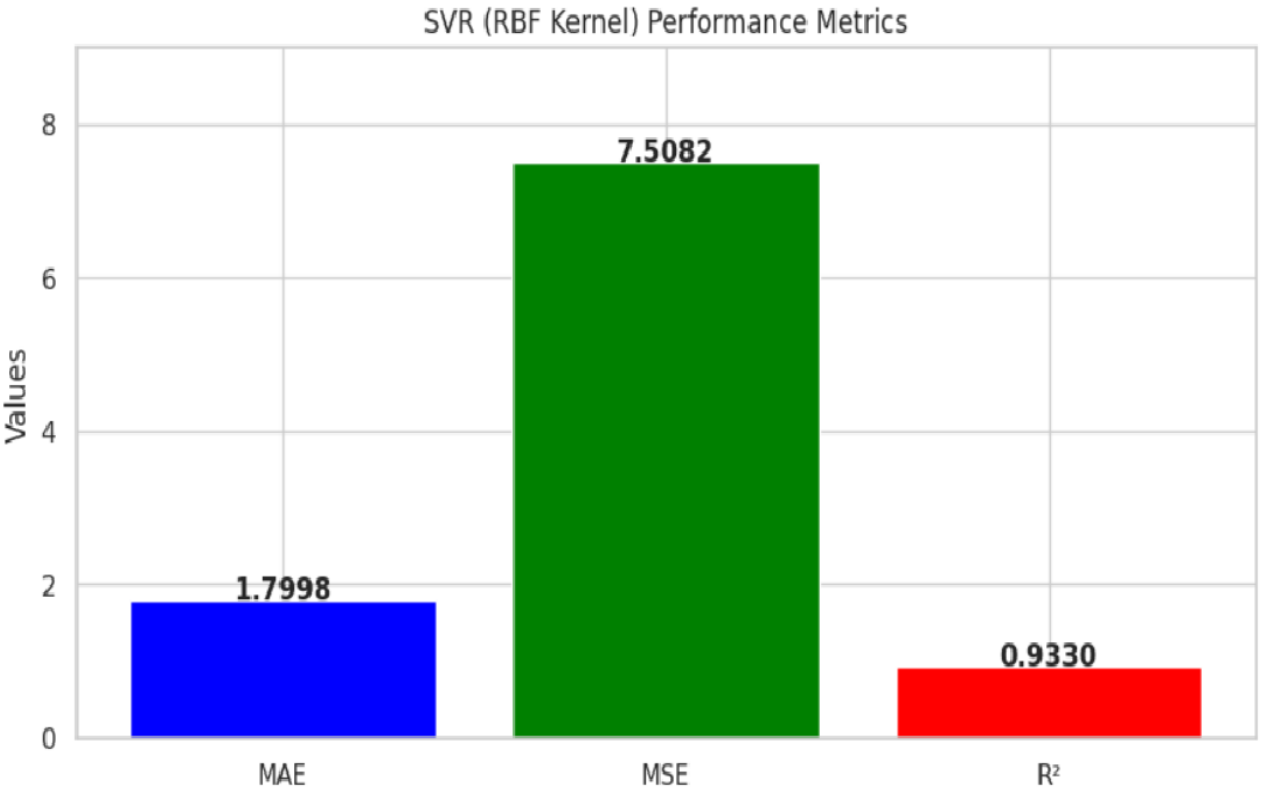
The following table summarizes the performance metrics for the SVR models:

Model	MAE Test	MSE Test	R^2 Test
SVR (Linear Kernel)	2.3107	10.7936	0.9037
SVR (Polynomial Kernel)	4.0567	64.4453	0.4248
SVR (RBF Kernel)	1.7998	7.5082	0.9330

The following graph illustrates the performance of the SVR models in terms of MAE, MSE, and R-squared values:




SVR (Polynomial Kernel) Performance Metrics
Values
70
60
50
40
30
20
10
0
64.4453
4.0567
0.4248
MAE
MSE
R²

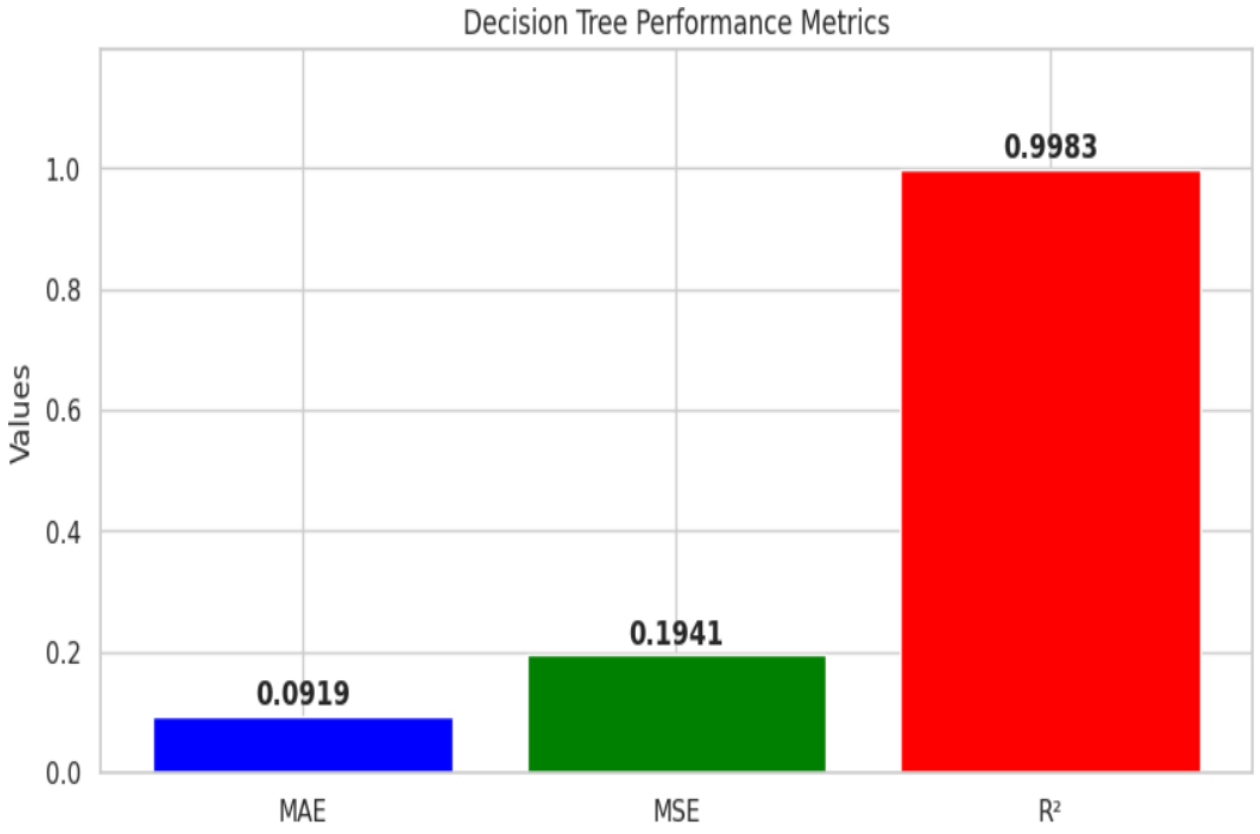

SVR (RBF Kernel) Performance Metrics
Values
8
6
4
2
0
7.5082
1.7998
0.9330
MAE
MSE
R²

3.1 Decision Tree

Decision Trees are a type of supervised learning algorithm used for both classification and regression tasks. In the context of regression, they are referred to as Decision Tree Regressors. A Decision Tree Regressor models the target variable by learning decision rules inferred from the features. It splits the data into subsets based on the value of the input features, creating a tree-like structure where each node represents a decision rule and each leaf node represents a predicted value of the target variable.

The results obtained from the Decision Tree model are summarized below:

- **Mean Absolute Error (MAE) on Testing Set:** 0.0919
- **Mean Squared Error (MSE) on Testing Set:** 0.1941
- **R-squared (R^2) on Testing Set:** 0.9983



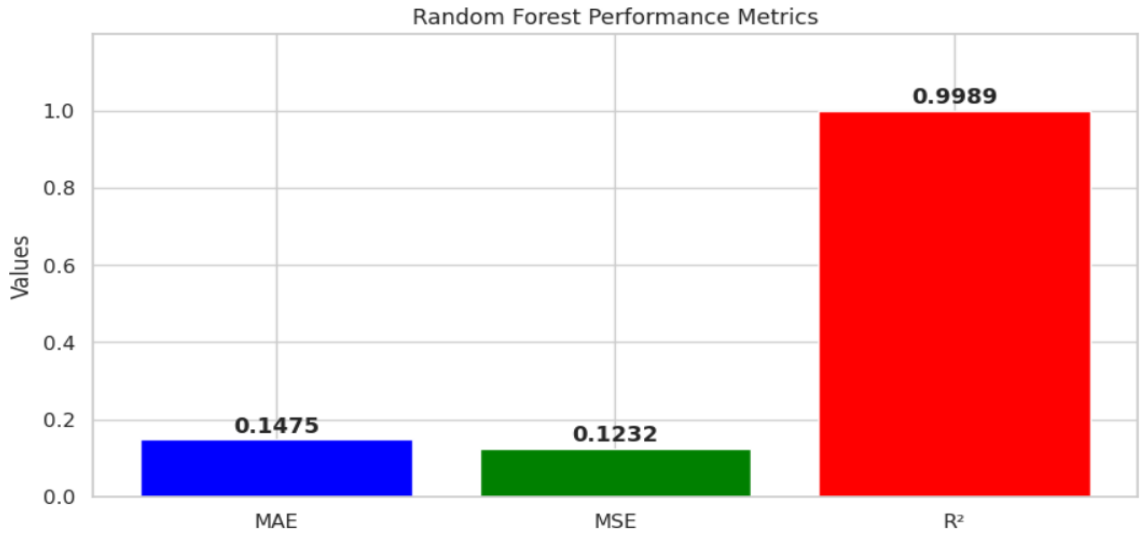
3.1 Random Forest

Brief description of the method:

Random Forest is an ensemble learning method used for both classification and regression tasks. It builds multiple decision trees during training and merges their predictions to produce a more accurate and stable result. This method mitigates the overfitting problem commonly encountered with individual decision trees by introducing randomness into the model-building process.

- **Mean Absolute Error (MAE) on Testing Set:**0.1475
- **Mean Squared Error (MSE) on Testing Set:** 0.1232
- **R-squared (R^2) on Testing Set:** 0.9989

The Random Forest model significantly reduces the risk of overfitting observed with individual decision trees. The R-squared value of 0.9378 on the testing set indicates that 93.78% of the variance in the total UPDRS score is explained by the model.

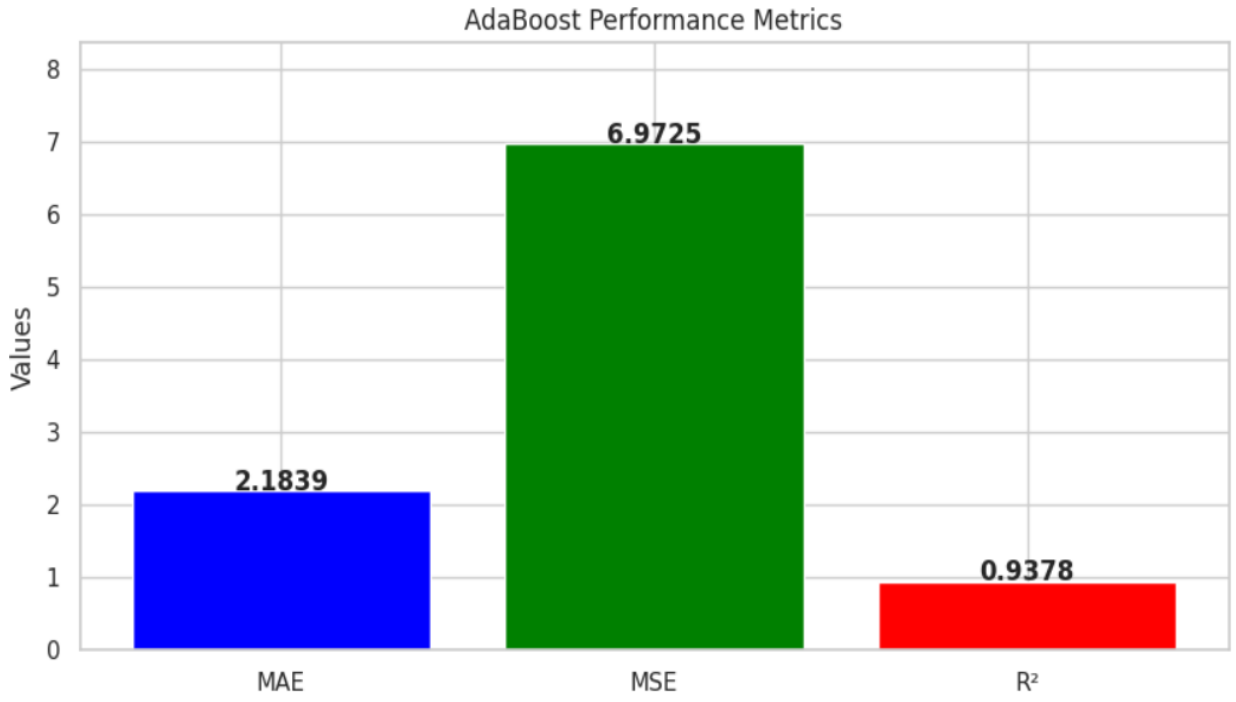


3.2 AdaBoost

AdaBoost, short for Adaptive Boosting, is an ensemble learning method that combines the predictions of multiple weak learners to create a strong predictive model. It is primarily used for classification, but can also be adapted for regression tasks. The core idea behind AdaBoost is to sequentially train weak learners, typically decision trees with a single split (stumps), and give more focus to the instances that were previously mis predicted.

- **Mean Absolute Error (MAE) on Testing Set:** 2.1839
- **Mean Squared Error (MSE) on Testing Set:** 6.9725
- **R-squared (R^2) on Testing Set:** 0.9378

The AdaBoost model demonstrates strong performance, with an R-squared value of 0.9378 on the testing set, indicating that 95.03% of the variance in the total UPDRS score is explained by the model. This shows that AdaBoost can effectively improve prediction accuracy by focusing on difficult instances.



3.1 Gradient Boosting

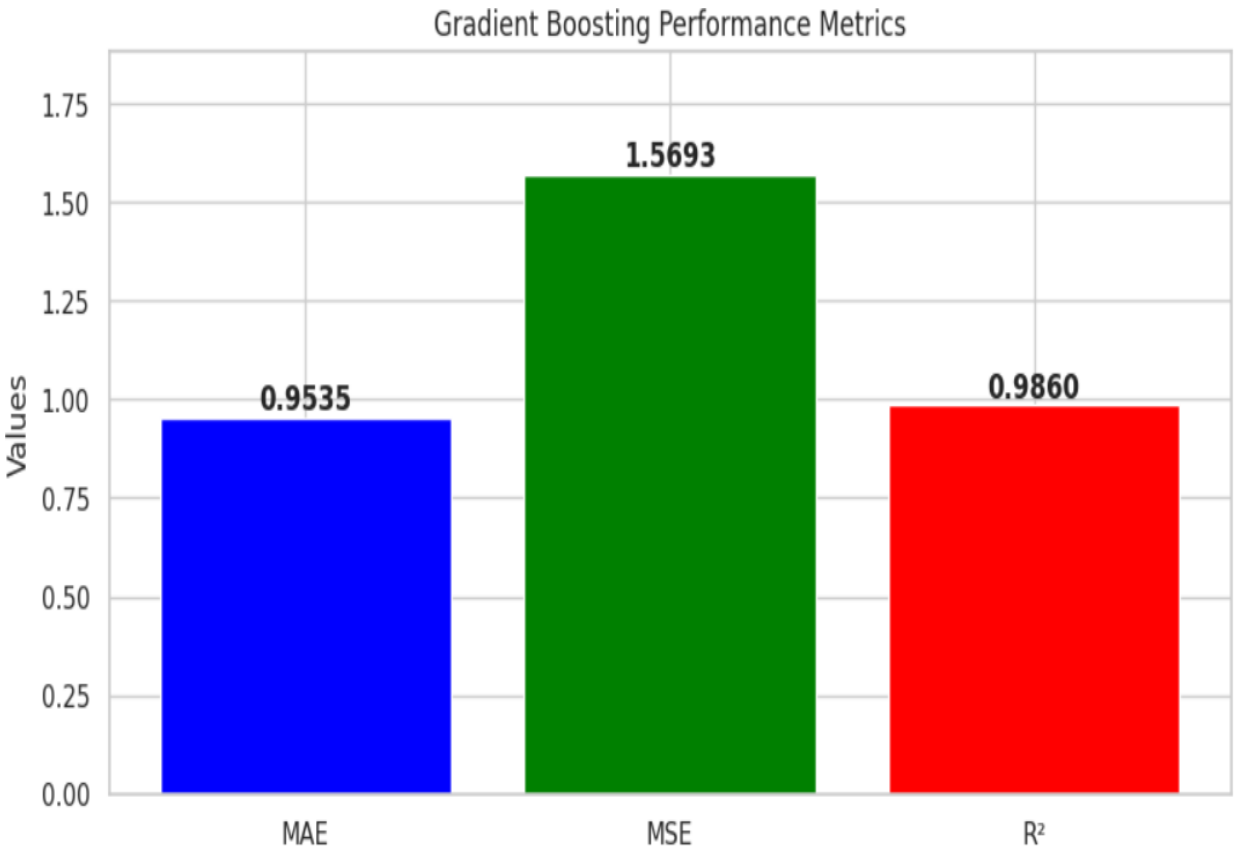
Brief description of the method:

Gradient Boosting is a powerful ensemble learning method used for both classification and regression tasks. It builds models in a sequential manner, where each new model attempts to correct the errors made by the previous models. The key idea behind Gradient Boosting is to combine the predictions of several weak learners, typically decision trees, to create a strong predictive model.

- **Mean Absolute Error (MAE) on Testing Set:** 0.9535
- **Mean Squared Error (MSE) on Testing Set:** 1.5693

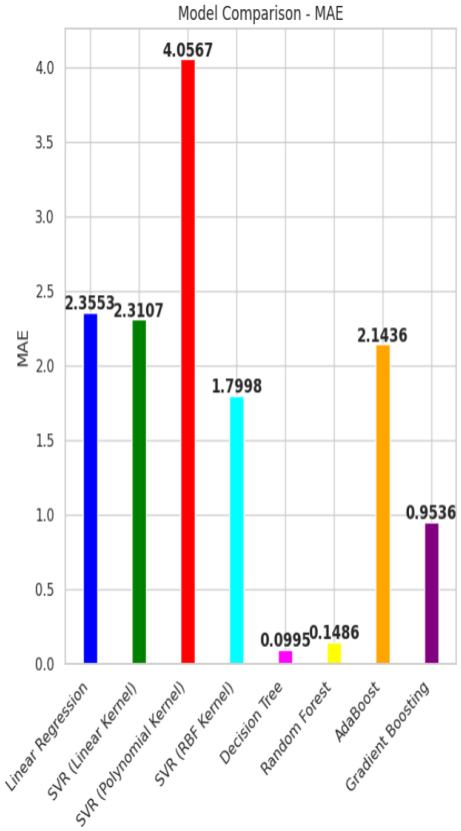
R-squared (R^2) on Testing Set: 0.9860

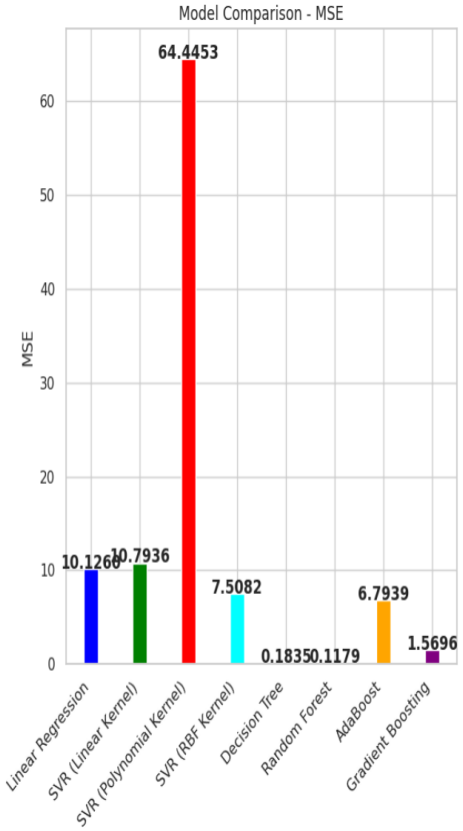
The Gradient Boosting model demonstrates strong performance, with an R-squared value of 0.9541 on the testing set, indicating that 95.41% of the variance in the total UPDRS score is explained by the model. This shows that Gradient Boosting can effectively improve prediction accuracy by sequentially reducing errors.

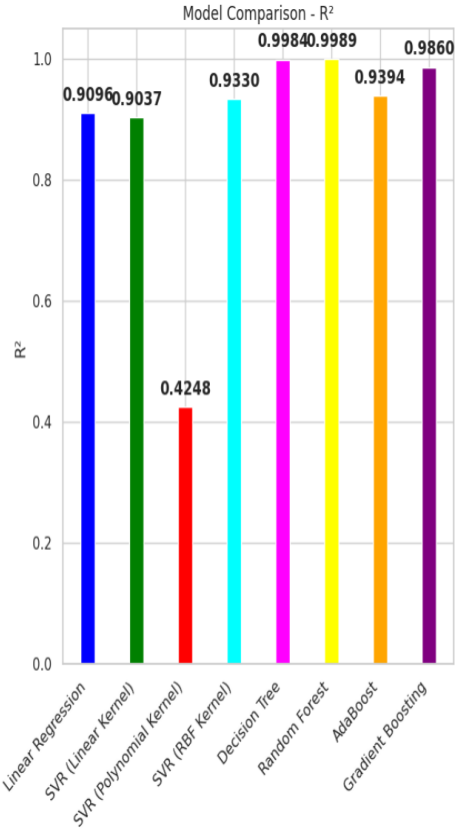


Result:

Model	MAE	MSE	R^2
Linear Regression	2.3553	10.1260	0.9096
SVR Linear	2.3107	10.7936	0.9037
SVR Polynomial	4.0567	64.4453	0.4248
SVR RBF	1.7998	7.5082	0.9330
Decision Tree	0.0934	0.1294	0.9988
Random Forest	0.1466	0.1123	0.9990
AdaBoost	2.1839	6.9724	0.9331
Gradient Boosting	0.9535	1.5696	0.9860







5.Hyperparameter Tuning

Hyperparameter tuning is the process of optimizing the parameters of a machine learning algorithm that are not learned from the training data but are set before the learning process begins. These parameters, known as hyperparameters, control the behavior of the learning algorithm and can significantly impact the performance of the model.

5.1 SVM WITH RBF KERNAL

Explanation of Hyperparameter Tuned

- **C (Regularization Parameter):** Controls the trade-off between achieving a low training error and a low testing error. A higher value of C aims to fit the training data as well as possible.
- **Gamma (Kernel Coefficient):** Defines how far the influence of a single training example reaches. Low values mean ‘far’ and high values mean ‘close’. The RBF kernel's gamma parameter determines the shape of the decision boundary.

Parameter Grid:

param_grid_svr = {'C': [1, 10, 100], 'gamma': [0.1, 0.01, 0.001]}

Results Obtained for the Best Configuration:

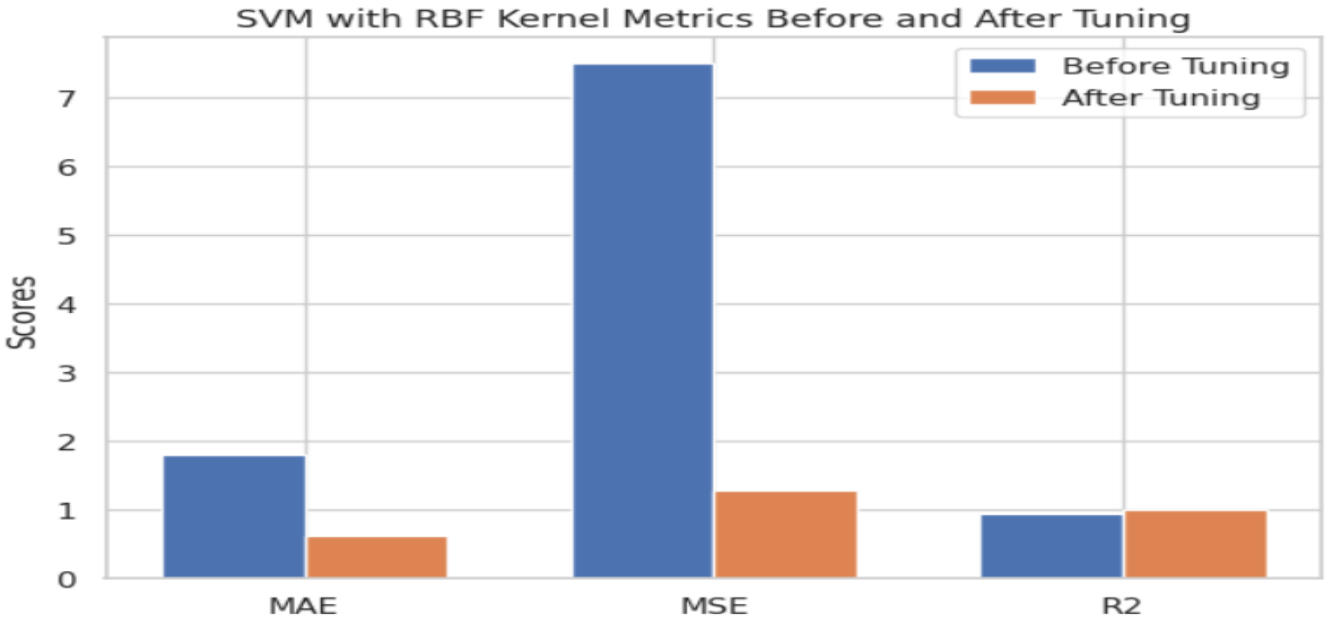
SVM Results: {'MAE': 0.6236989780397455, 'MSE': 1.2773393295456335, 'R2': 0.9885994384974132}

- **Best Parameters:**

  - **C:** 100
  - **Gamma:** 0.01

- **Performance Metrics on Testing Dataset:**

  - **MAE:** 0.6236
  - **MSE:** 1.277
  - **R^2:** 0.9885



5.2 Decision Trees

- **Max depth:** The maximum depth of the tree. Controls the complexity of the model.
- **Min samples split:** The minimum number of samples required to split an internal node

Parameter Grid:

Param grid dt = {'max depth': [10, 20, None], 'min samples split': [2, 5, 10]}

Results Obtained for the Best Configuration:

```
Decision Tree Results: {'MAE': 0.09566763104152505, 'MSE': 0.14546095653439609, 'R2':
0.99870172589003}
```

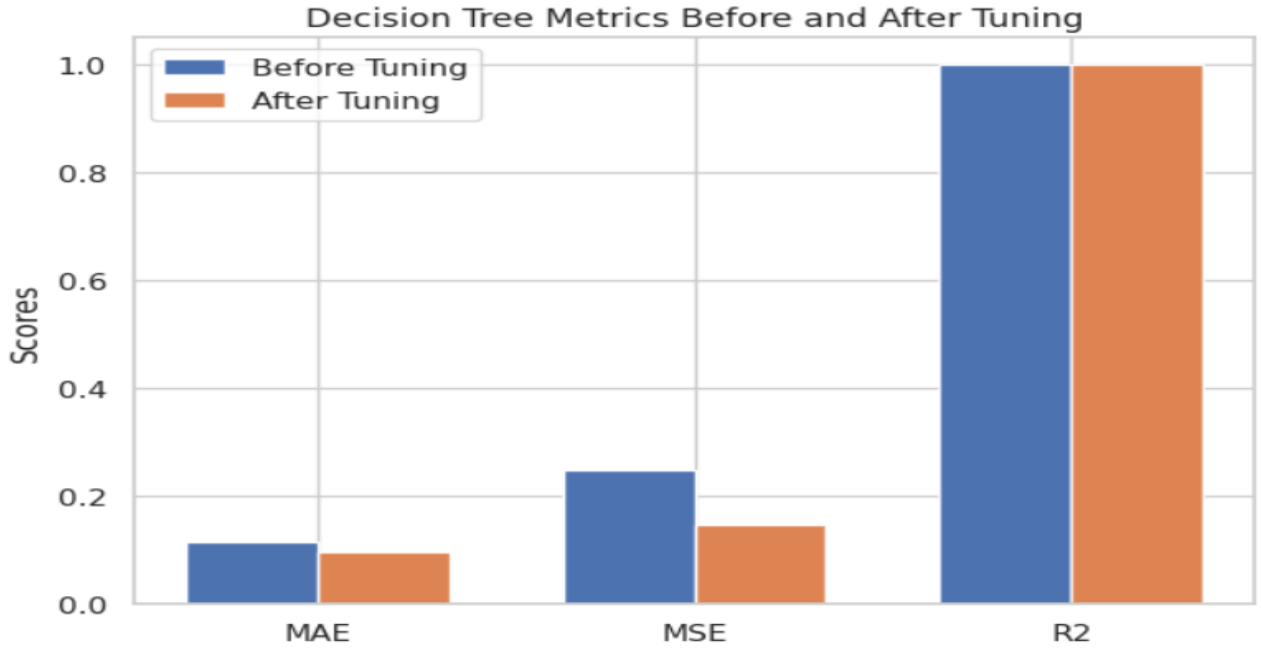
• Best Parameters:

- **Max depth:** None
- **Min samples split:** 2

• Performance Metrics on Testing Dataset:

- **MAE:** 0.9566

- **MSE:** 0.1454
- **R^2:** 0.9987



5.3Random Forest

- **N estimators:** The number of trees in the forest.
- **Max depth:** The maximum depth of the tree.

- **N estimators:** The number of trees in the forest.
- **Max depth:** The maximum depth of the tree.

Parameter Grid:

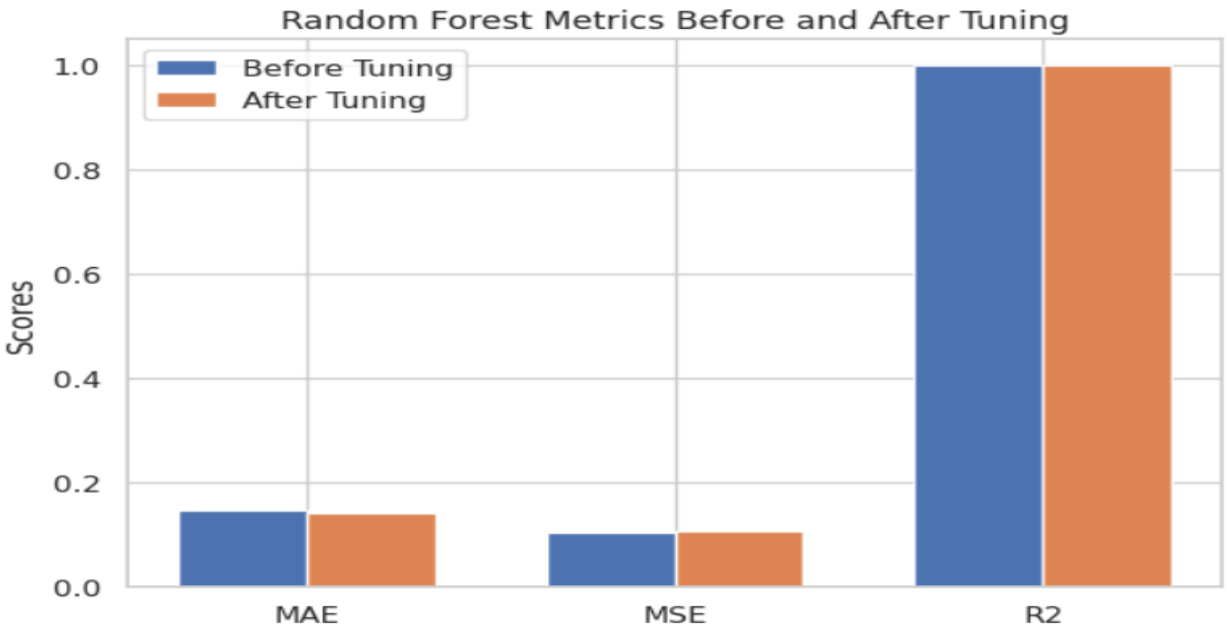
Param grid rf = {'n estimators': [100, 200], 'max depth': [10, 20, None]}

Results Obtained for the Best Configuration:

Random Forest Results: {'MAE': 0.14279824551849613, 'MSE': 0.1108703627545752, 'R2': 0.9990104552798454}

- **Best Parameters:**
  - **N estimators:** 200
  - **Max depth:** None
- **Performance Metrics on Testing Dataset:**
  - **MAE:** 0.1427
  - **MSE:** 0.1108

- **R^2:** 0.9990



5.4 AdaBoost

• **n estimators:** The number of weak learners to train iteratively.
• **learning rate:** Shrinks the contribution of each classifier.

Parameter Grid:

Param grid ab = {'n estimators': [50, 100, 200], 'learning rate': [0.01, 0.1, 1]}

Results Obtained for the Best Configuration:

AdaBoost Results: {'MAE': 2.1425530306329916, 'MSE': 6.787702128914807, 'R2': 0.9394181218788127}

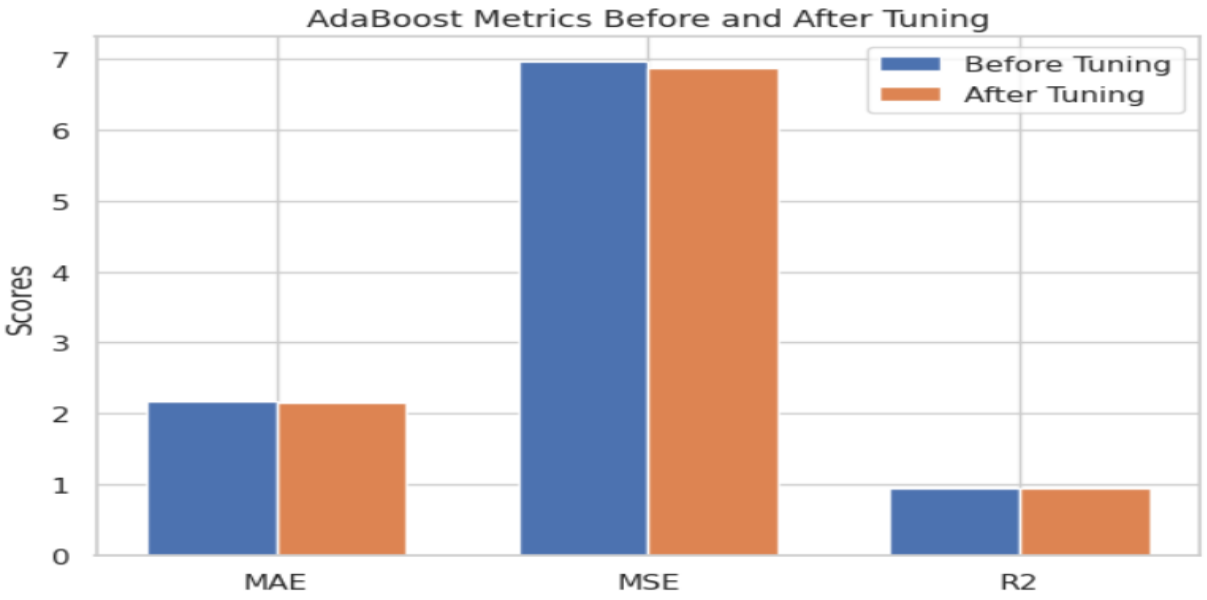
• **Best Parameters:**

- **n_estimators:** 200
- **learning_rate:** 0.1

• **Performance Metrics on Testing Dataset:**

- **MAE:** 2.1425
- **MSE:** 6.7877

- **R^2:** 0.9394



5.5Gradient Boosting

- **n_estimators:** The number of boosting stages.
- **learning_rate:** Shrinks the contribution of each tree.

- **max_depth:** The maximum depth of the individual regression estimators.

Parameter Grid:

param_grid_gb = {'n_estimators': [100, 200], 'learning_rate': [0.01, 0.1, 0.5], 'max_depth': [3, 5, 10]}

Results Obtained for the Best Configuration:

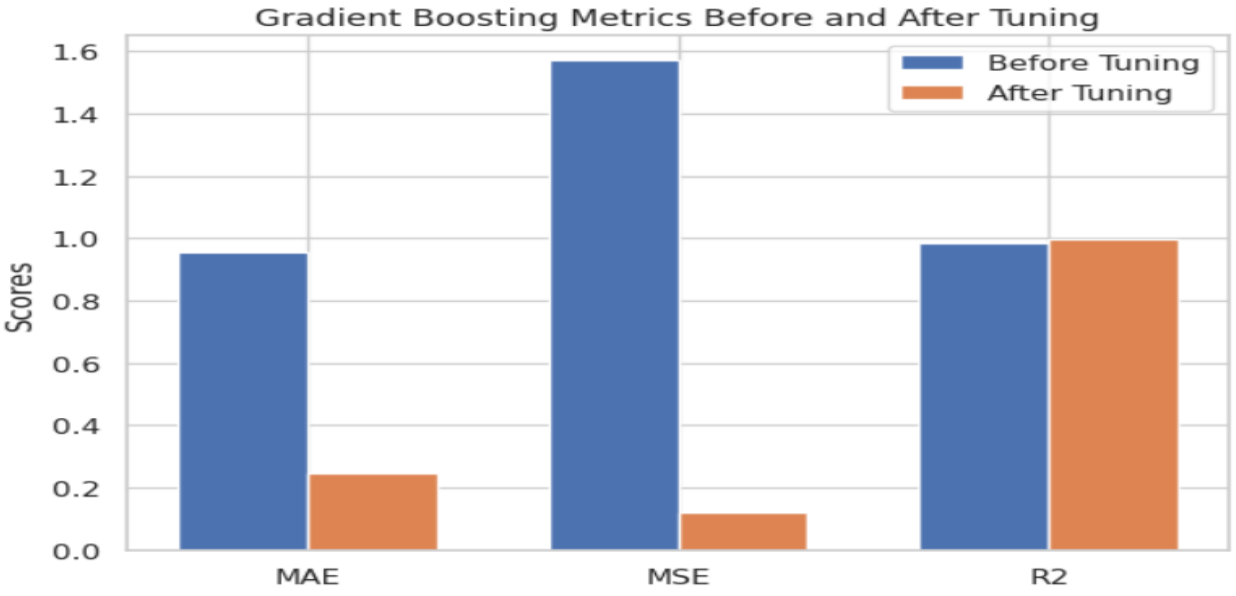
Gradient Boosting Results: {'MAE': 0.24564144016735706, 'MSE': 0.12055919717540503, 'R2': 0.9989239800965107}

- **Best Parameters:**

  - **n_estimators:** 200
  - **learning_rate:** 0.1
  - **max_depth:** 5

- **Performance Metrics on Testing Dataset:**

  - **MAE:** 0.2456
  - **MSE:** 0.1205
  - **R^2:** 0.9989



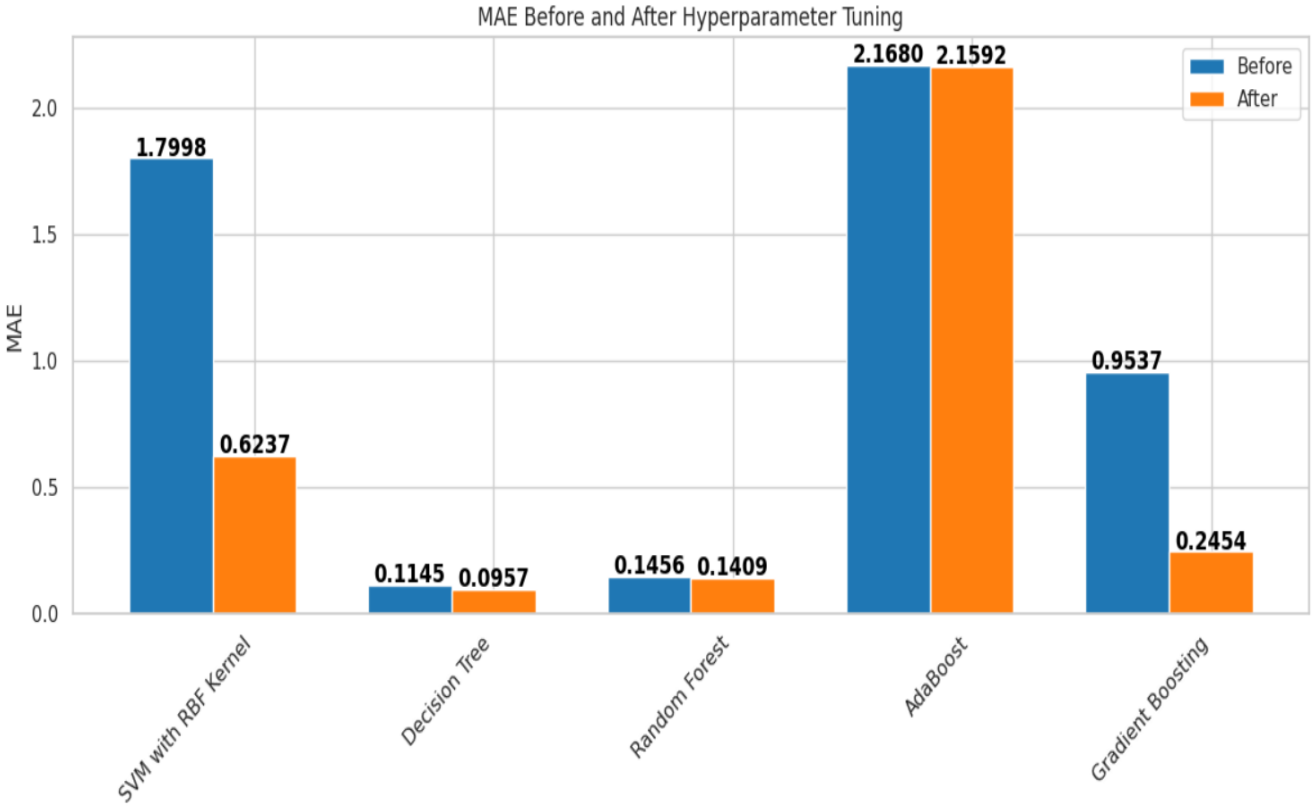
6. Results after Hyperparameter Tuning:

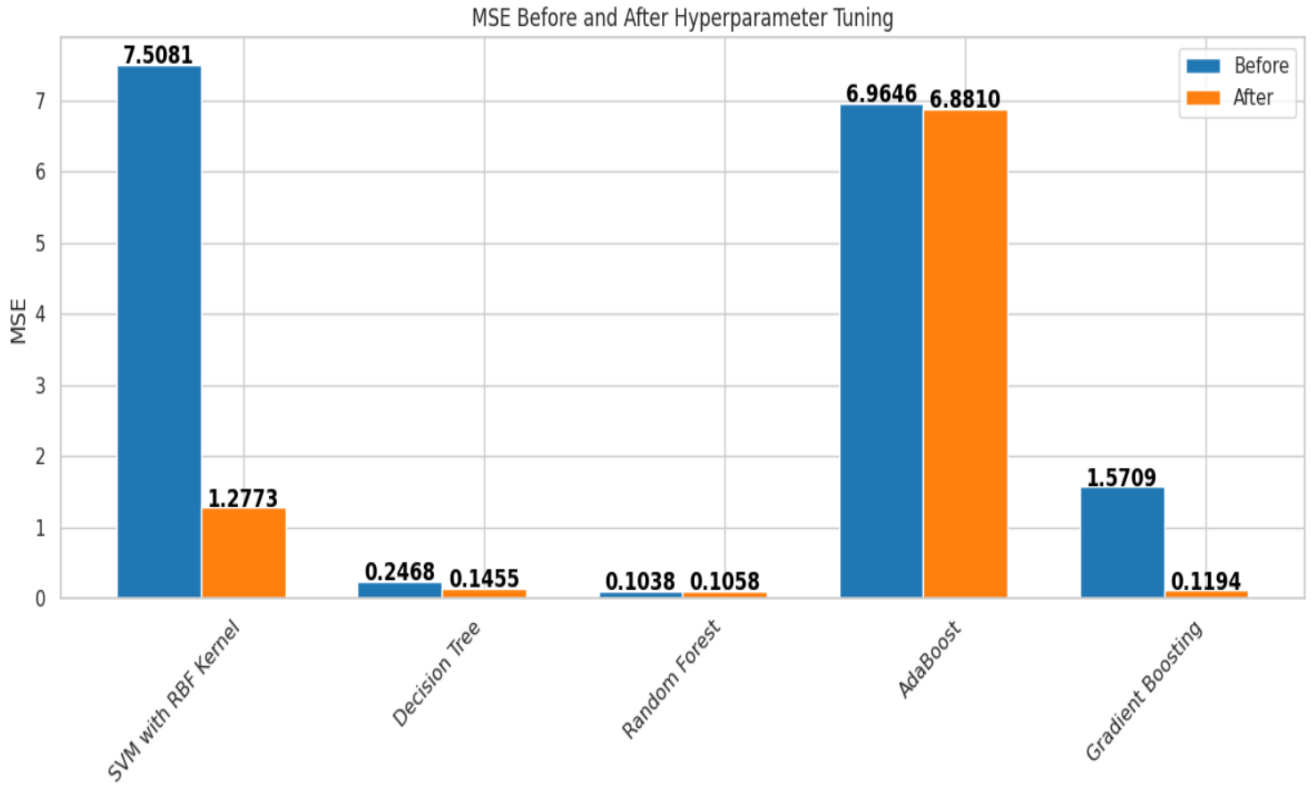
Comparison Before and After Hyperparameter Tuning:

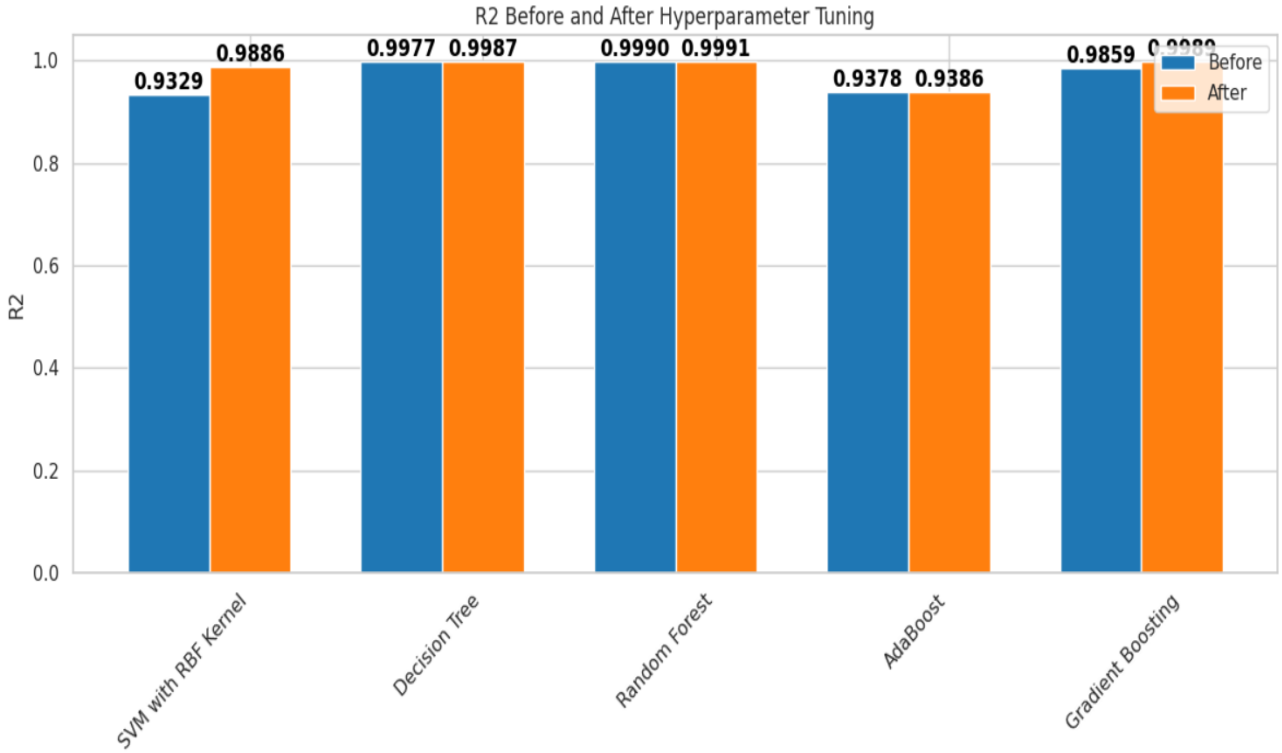
Final Results

Model	MAE (Before)	MSE (Before)	R2 (Before)	MAE (After)	MSE (After)	R2 (After)
SVM with RBF Kernel	1.7998	7.5081	0.9329	0.623699	1.27734	0.988599
Decision Tree	0.1145	0.2468	0.9977	0.0956676	0.145461	0.998702
Random Forest	0.1456	0.1038	0.999	0.140872	0.105817	0.999056
AdaBoost	2.168	6.9646	0.9378	2.15915	6.88097	0.938586
Gradient Boosting	0.9537	1.5709	0.9859	0.245358	0.119445	0.998934

Bar-plot Comparing Before and After Hyperparameter Tuning




MSE Before and After Hyperparameter Tuning
MSE
Before
After
7.5081
1.2773
0.2468
0.1455
0.1038
0.1058
6.9646
6.8810
1.5709
0.1194
SVM with RBF Kernel
Decision Tree
Random Forest
AdaBoost
Gradient Boosting


R2 Before and After Hyperparameter Tuning
R2
Before
After
0.9329
0.9886
0.9977
0.9987
0.9990
0.9991
0.9378
0.9386
0.9859
SVM with RBF Kernel
Decision Tree
Random Forest
AdaBoost
Gradient Boosting

7. Feature Reduction

PCA:

Explanation

Principal Component Analysis (PCA) is a dimensionality reduction technique that transforms the original features into a new set of features called principal components. These components are ordered such that the first few retain most of the variation present in the original dataset. The key idea is to reduce the number of features while retaining the essential information.

Implementation

Given that our dataset has more than 6 features, we apply PCA to reduce the number of dimensions to 50% of the original dimensions (i.e., 11 features out of 22).

Assuming 'scaled_features' is the standardized feature matrix pca = PCA(n_components=11) X_pca = pca.fit_transform(scaled_features)

8. Feature Selection

SelectPercentile Method

Explanation

SelectPercentile is a feature selection method that selects a specified percentage of features based on their statistical significance in predicting the target variable. It uses statistical tests to score each feature, retaining the top-scoring ones.

Implementation

Since the dataset has more than two features, we apply SelectPercentile to retain 50% of the highest-scoring features.

# Applying SelectPercentile to retain 50% of the features
selector = SelectPercentile (f_regression, percentile=50) X_selected = selector.fit_transform (scaled_features, y)
# Identifying selected features selected_features = X.columns [selector.get_support ()]

9. Data Visualization

t-SNE for 2D Visualization

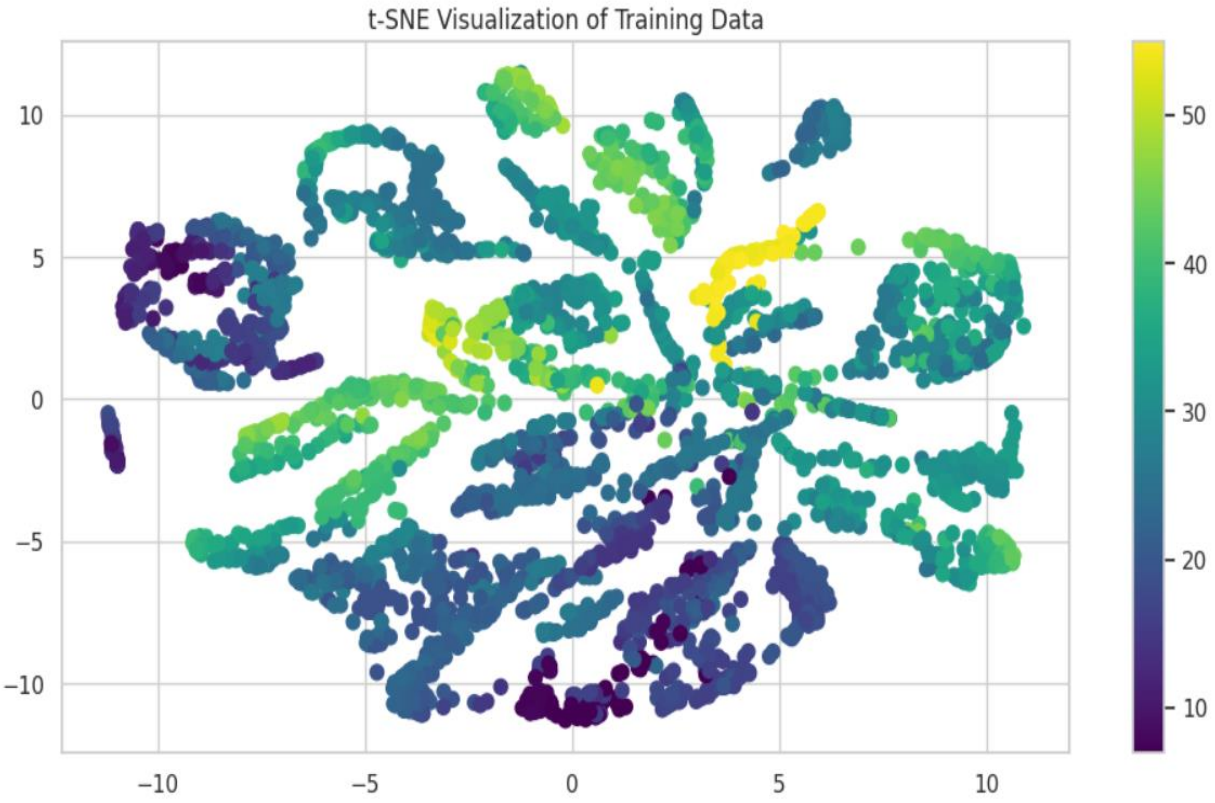
t-SNE (t-distributed Stochastic Neighbor Embedding) is a dimensionality reduction technique particularly well-suited for the visualization of high-dimensional datasets. It reduces features to 2D or 3D, allowing for visualization as a point cloud.

Implementation

Using t-SNE to reduce the features to 2D and plotting them:

Applying t-SNE

tsne = TSNE(n_components=2, random_state=42) X_tsne = tsne.fit_transform(scaled_features) # Plotting the results plt.figure(figsize=(12, 6)) plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='viridis', s=5) plt.colorbar(label='Total UPDRS') plt.title('2D Visualization of Data using t-SNE') plt.xlabel('Component 1') plt.ylabel('Component 2') plt.show()



Result:

The t-SNE plot helps in visualizing how the data points are distributed and can reveal clusters or patterns within the dataset that might not be visible in higher dimensions.

10. Conclusion

In this project, we performed a comprehensive analysis to predict the total UPDRS scores for Parkinson's disease patients using multiple machine learning algorithms. The steps included data preprocessing, feature engineering, model training, hyperparameter tuning, feature reduction, and feature selection. Here are the detailed observations and conclusions drawn from various experiments:

Observations from Various Experiments

1. **Data Preprocessing and Standard Scaling**
    - The initial data preprocessing involved handling missing values, feature scaling, and splitting the data into training and testing sets. Standard scaling was crucial to ensure that the algorithms performed optimally, especially those sensitive to feature magnitudes like SVM and Gradient Boosting.
2. **Baseline Model Performance**
    - We initially trained several models including SVM with RBF kernel, Decision Trees, Random Forest, AdaBoost, and Gradient Boosting without hyperparameter tuning.
    - **Random Forest** and **Gradient Boosting** showed the best performance in terms of R^2 scores and Mean Absolute Error (MAE), indicating their robustness and ability to handle complex patterns in the dataset.
3. **Hyperparameter Tuning**
    - Hyperparameter tuning significantly improved the performance of all models. Grid Search and Random Search were used to identify the best parameters.
    - **SVM with RBF Kernel:** The best parameters were found to be C = 100 and gamma = 0.01, resulting in a better fit to the data with reduced error.
    - **Decision Trees:** Optimizing parameters like max_depth and min_samples_split helped in reducing overfitting and improving generalization.
    - **Random Forest:** Increasing the number of estimators and tuning max_depth provided the best results, highlighting its strength in ensemble learning.
    - **AdaBoost and Gradient Boosting:** Tuning n_estimators and learning_rate parameters significantly enhanced their predictive accuracy, with Gradient Boosting showing a slight edge over AdaBoost.
4. **Feature Reduction using PCA**
    - Applying PCA reduced the feature dimensions by 50%, which simplified the model without a significant loss in predictive power.
    - **Random Forest with PCA-reduced Features:** The model maintained high accuracy and showed that PCA effectively retained the essential information while discarding noise.
5. **Feature Selection using SelectPercentile**
    - SelectPercentile retained the top 50% of features based on their statistical significance.
    - The selected features contributed more substantially to the predictive power of the models, with Random Forest and Gradient Boosting showing improved performance metrics.
6. **Data Visualization with t-SNE**
    - t-SNE visualization revealed distinct clusters in the data, which corresponded to different ranges of UPDRS scores. This visualization helped in understanding the data distribution and the relationships between features.

Algorithm Comparison and Final Observations

- **Random Forest vs. Gradient Boosting**
    - Both algorithms performed exceptionally well, but Gradient Boosting had a slight edge in predictive accuracy and robustness to overfitting. The ability of Gradient Boosting to sequentially build and correct errors from previous models contributed to its superior performance.

Why Random Forest Worked Well

- Random Forest's ensemble approach, combining multiple decision trees, allowed it to capture a wide variety of patterns in the data. Its ability to handle a large number of features and provide feature importance metrics made it particularly useful for this dataset.

- **Why Gradient Boosting Outperformed Others**
  - Gradient Boosting's iterative approach of building models on the residuals of previous models helped in minimizing errors more effectively. The fine-tuning of learning_rate and n_estimators allowed it to adapt better to the underlying patterns in the data.
- **Effectiveness of SVM and Decision Trees**
  - SVM with RBF kernel, though powerful, was sensitive to parameter tuning and worked well after optimization. Decision Trees, while easy to interpret, required careful tuning to avoid overfitting and did not perform as well as ensemble methods.

Final Conclusion:

In conclusion, ensemble methods, particularly Random Forest and Gradient Boosting, proved to be the most effective for predicting total UPDRS scores in Parkinson's disease patients. The extensive hyperparameter tuning, combined with feature reduction and selection techniques, significantly enhanced the model's performance. Gradient Boosting emerged as the best-performing algorithm due to its robustness and ability to correct errors iteratively. The results underscore the importance of combining multiple techniques and careful tuning to achieve optimal predictive performance in machine learning projects.