Comparative Analysis of Machine Learning Models for Obesity Level Prediction: A Study using Decision Tree, Random Forest, and SVM

Ayush Rayamajhi
Data Science and Computational Intelligence
Softwarica College Of IT and E-Commerce
Kathmandu, Nepal
Coventry University
240178@softwarica.edu.np

***Abstract*—This coursework involves developing and analyzing machine learning models—Decision Tree, Random Forest, and Support Vector Machine (SVM)—to predict obesity levels using a dataset retrieved from the UCI repository. Feature selection was primarily performed using Mutual Information to reduce model complexity, further refined through insights from Exploratory Data Analysis (EDA). A Random Forest model was integrated into a user-friendly Streamlit UI, enabling real-time obesity predictions based on user input.**

The models were evaluated using metrics such as accuracy, confusion matrices, ROC curves, and cross-validation. EDA techniques, including histograms and density plots, offered insights into the dataset, leading to model improvements. This study emphasizes the predictive power of Random Forest and demonstrates the practical application of machine learning in healthcare.

***Index Terms*—Machine Learning, Obesity Prediction, Decision Tree, Random Forest, Support Vector Machine, Feature Selection, Streamlit UI, Healthcare Data**

I. Introduction

Obesity is a significant medical condition linked to chronic diseases such as diabetes and cardiovascular diseases [1]. Early identification of risk for obesity is critical for preventing such diseases [2]. In this coursework, machine learning models—Decision Tree, Random Forest, and Support Vector Machine (SVM)—are developed to predict obesity levels using a dataset obtained from UCI website [3]. Mutual Information is applied for primary feature selection and to reduce complexity of the model [4], and the model is further enhanced using the information obtained from EDA [8].

A Random Forest model is deployed through a Streamlit interface [6], enabling users to input health data and receive real-time predictions of obesity risk. The models are evaluated using accuracy, confusion matrices, ROC curves, and cross-validation [7]. Additionally, EDA is used to understand feature distributions and their impact on predictions [8].

A comparative analysis is conducted between the three models to evaluate their performance and highlight the strengths and weaknesses of each model. This project demonstrates the practical use of machine learning in healthcare, emphasizing the importance of feature selection and evaluation in building effective predictive models.

II. Literature review

The research paper ”Obesity Level Estimation Software based on Decision Trees” by Eduardo De-La-Hoz-Correa et al. explores machine learning techniques for estimating obesity levels [9]. The dataset includes 712 records from young adults in Colombia, Mexico, and Peru, with factors such as gender, age, weight, height, physical activity, and fast food consumption. The study applies the SEMMA methodology (sampling, exploring, modifying, modeling, and assessing) to classify obesity based on WHO’s BMI standards.

Three models—Decision Trees, Naïve Bayes, and Logistic Regression—were tested. Decision Trees achieved the highest precision (97.4%), making it the best-performing model for the software [10].

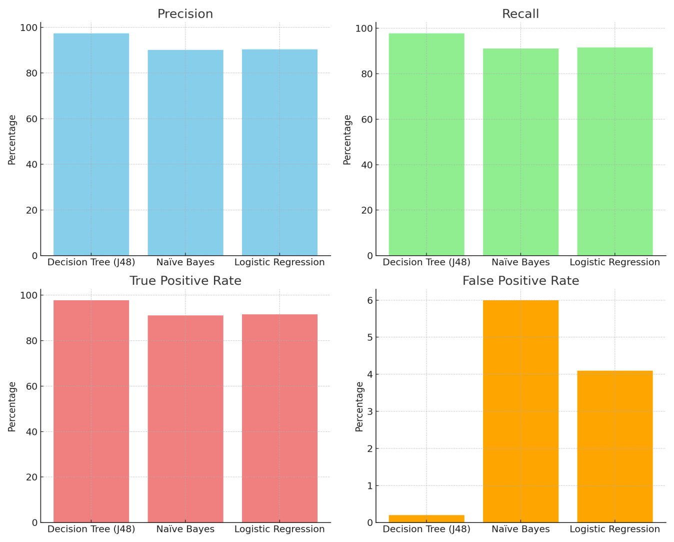


Fig. 1: Performance Comparison of Machine Learning Models for Obesity Level Prediction

The plot compares these models on precision, recall, TP rate, and FP rate. The Decision Tree outperforms the others, with the highest precision (97.4%) and recall (97.8%) and the lowest FP rate (0.2%). Naïve Bayes and Logistic Regression show lower performance, with Naïve Bayes having the highest FP rate (6.0%) [12].

In conclusion, this study enhances the accuracy of obesity prediction tools and demonstrates the effectiveness of machine learning in addressing public health issues like obesity.

III. Dataset Description

The dataset used in this coursework was retrieved from the UCI Machine Learning Repository [13]. It contains 2,111 records of individuals from Mexico, Peru, and Colombia, aged between 14 and 61. The dataset is designed for estimating obesity levels based on eating habits and physical conditions. Notably, 77% of the data was synthetically generated using SMOTE by the publishers of the dataset [14], while 23% consists of real data collected via a web-based survey [15].

A. Features

The dataset consists of 17 attributes that describe various aspects of individuals' eating habits, physical activities, and demographics. The target variable is obesity level, which is categorized into seven distinct classes based on the World Health Organization (WHO) and Mexican Normativity guidelines. Below is a summary of the features:

- **Gender**: Categorical, indicating Male or Female.
- **Age**: Numerical, age of the individual in years.
- **Height**: Numerical, height of the individual in meters.
- **Weight**: Numerical, weight of the individual in kilograms.
- **family_history_with_overweight**: Categorical, indicating whether a family member has suffered or currently suffers from being overweight (Yes/No).
- **FAVC (Frequent consumption of high-caloric food)**: Categorical, indicating frequent intake of high-caloric foods (Yes/No).
- **FCVC (Frequency of consumption of vegetables)**: Categorical, describing how often the individual consumes vegetables (Never, Sometimes, Always).
- **NCP (Number of main meals per day)**: Categorical, indicating the number of main meals (1-2, Three, More than Three).
- **CAEC (Consumption of food between meals)**: Categorical, indicating whether the individual eats between meals (No, Sometimes, Frequently, Always).
- **SMOKE**: Categorical, indicating whether the individual smokes (Yes/No).
- **CH2O (Daily water intake)**: Categorical, describing daily water consumption (Less than 1 liter, 1-2 liters, More than 2 liters).
- **SCC (Calories consumption monitoring)**: Categorical, indicating whether the individual monitors their calorie intake (Yes/No).
- **FAF (Frequency of physical activity)**: Categorical, indicating how often the individual engages in physical activity (None, 1-2 days, 2-4 days, 4-5 days).
- **TUE (Time spent on technology)**: Categorical, indicating time spent on technology devices such as phones, computers, and televisions (0-2 hours, 3-5 hours, More than 5 hours).
- **CALC (Alcohol consumption)**: Categorical, indicating the frequency of alcohol consumption (I do not drink, Sometimes, Frequently, Always).
- **MTRANS (Transportation used)**: Categorical, indicating the primary mode of transportation (Automobile, Motorbike, Bike, Public Transport, Walking).

B. Target Variable (Obesity Level)

The target variable, **NObesity**, categorizes individuals into one of seven classes based on their BMI (Body Mass Index), following WHO and Mexican Normativity standards:

- Insufficient Weight
- Normal Weight
- Overweight Level I
- Overweight Level II
- Obesity Type I
- Obesity Type II
- Obesity Type III

The dataset provides a valuable resource for developing machine learning models to predict obesity levels, as it covers a range of factors influencing an individual's weight and physical condition [15].

IV. Methods

In this section, we developed three machine learning models—Decision Tree, Random Forest, and Support Vector Machine (SVM)—to predict obesity levels based on individuals' health and lifestyle features. The primary feature selection method used was **Mutual Information**, which helped reduce the feature space and improved model performance by focusing on the most relevant features.

- **Decision Tree**: The Decision Tree is a simple yet powerful classification algorithm that recursively splits the dataset into smaller subsets based on the most informative features [16]. This process continues until the data is split into pure or nearly pure subsets. In this project, key features such as Weight, Height, and Age were selected after preprocessing and feature selection.

  The splitting criterion used in this Decision Tree is based on Gini Impurity, which measures how often a randomly chosen element from the set would be incorrectly classified if it was randomly labeled according to the distribution of labels in the subset. The goal of the Decision Tree algorithm is to minimize this impurity at each split [19].

  The formula for Gini Impurity is:

$$Gini(p) = 1 - \sum_{i=1}^{n} p_i^2$$

Where:

- p_i is the proportion of class i in the node.
- n is the total number of classes.

At each step of the tree-building process, the algorithm evaluates all possible features and thresholds, selecting the one that results in the lowest Gini impurity after the split. This recursive process continues until a stopping criterion is met, such as reaching a maximum tree depth or achieving pure node splits.

- **Random Forest**: Random Forest is a powerful ensemble learning method that combines the predictions of multiple decision trees to produce a more accurate and robust result [17]. Each decision tree in the forest is trained on a randomly selected subset of the dataset, both in terms of data instances and features. This randomness reduces overfitting and increases the model's ability to generalize to unseen data [20].
  In Random Forest, each tree independently generates a prediction, and the final prediction is made by aggregating the outputs—either by taking the majority vote for classification tasks or by averaging the results for regression tasks. The randomness not only stems from the bootstrap sampling of the data but also from selecting a random subset of features at each split. This ensures that each tree is diverse and brings unique insights to the final ensemble model [17].

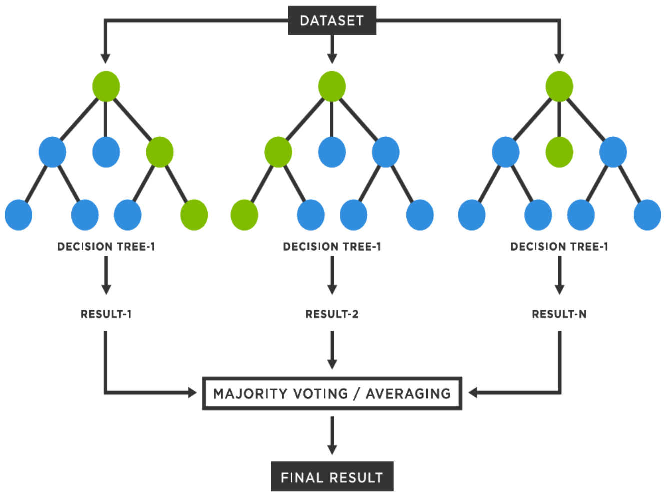


Fig. 2: Random Forest Model: Multiple decision trees trained on random subsets of the data, with the final prediction based on majority voting or averaging.

By combining the outputs of multiple trees, Random Forest mitigates the weaknesses of individual decision trees, such as their tendency to overfit the data [20]. Each tree may produce a slightly different prediction, but by aggregating these predictions, Random Forest produces a more accurate and stable outcome. The model's performance was rigorously evaluated using cross-validation and accuracy metrics, demonstrating its strong generalization to unseen data [17].

- **Support Vector Machine (SVM)**: A supervised learning algorithm with a **linear kernel** was employed for this classification task due to the simplicity and efficiency of the kernel in high-dimensional spaces [18]. Features were scaled, and the model was trained to find the optimal hyperplane that separated obesity levels [21].

  1) Decision Function (Linear SVM): In a linear Support Vector Machine (SVM), the decision boundary that separates the classes is represented by a hyperplane, defined by the decision function [22]:

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

Where:

- $\mathbf{x}$ is the input feature vector.
- $\mathbf{w}$ is the weight vector, which contains the coefficients learned during the training process.
- b is the bias term, which adjusts the position of the hyperplane.
- $\mathbf{w}^T\mathbf{x}$ is the dot product between the weight vector $\mathbf{w}$ and the input vector $\mathbf{x}$, representing the projection of $\mathbf{x}$ onto the direction of $\mathbf{w}$.

The classification is determined based on the sign of the decision function:

- If $f(\mathbf{x}) \geq 0$, the input $\mathbf{x}$ is classified as belonging to the positive class.
- If $f(\mathbf{x}) < 0$, the input $\mathbf{x}$ is classified as belonging to the negative class.

The goal of the SVM is to find the optimal hyperplane that maximizes the margin between the two classes, ensuring robust and accurate classification [22].

V. Experimental Setup

The development of the machine learning models involved several key steps, from data preprocessing to model training and evaluation. These steps ensured that the models were trained on clean, relevant data, and that their performance was thoroughly assessed [23]. The following outlines the process used to prepare the dataset, select features, train the models, and evaluate their accuracy and robustness [24].

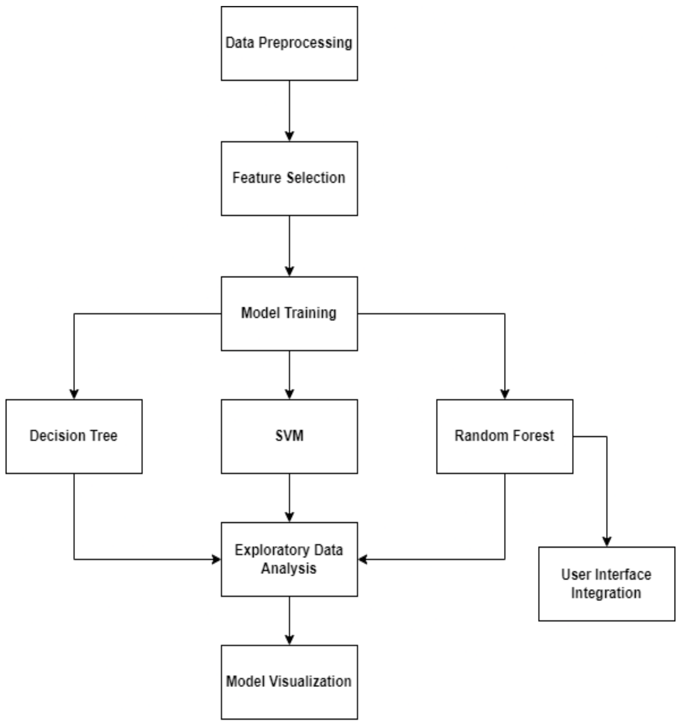


Fig. 3: Flowchart of the Development process of Models

Model evaluation was conducted using cross-validation, accuracy metrics, and confusion matrices to ensure robustness and generalization [25].

A. *Development Process*

The models were developed in the following steps:

- **Data Preprocessing**: Label encoding was applied to categorical features like *Gender* and *family_history_with_overweight*. The dataset was split into training (80%) and testing (20%) sets for model development and evaluation [26].
- **Feature Selection**: **Mutual Information** was applied to identify key features contributing to obesity prediction, and unnecessary features were dropped to simplify the models and enhance generalization. This process was followed consistently across all three models [26].
- **Model Training**: Each model was trained using the reduced feature set:
  - **Decision Tree** and **Random Forest**: Tree-based models were trained with hyperparameters optimized for accuracy and robustness.
  - **SVM**: A linear kernel was used with appropriate regularization to balance the classification task [27].
- **Model Evaluation**: The models were evaluated using metrics like **accuracy**, **confusion matrices**, **ROC curves**, and **cross-validation** [27]. The performance of each model was assessed, and the models' ability to predict obesity levels was compared. A **learning curve** was plotted to show how performance improved with increasing amounts of training data.
- **Exploratory Data Analysis (EDA)**: Post-training, **EDA** was conducted using histograms and density plots to visualize the distribution of key features like Weight, Height, and Age, providing insights into their contribution to obesity classification [28].
- **Model Visualization**: The **feature importance** values from Decision Tree and Random Forest models were visualized using bar charts, highlighting the most influential features in predicting obesity levels [28].
- **Streamlit UI Integration** (Random Forest only): The **Random Forest** model was integrated into a user-friendly **Streamlit UI** to allow real-time obesity level predictions based on user inputs [29]. This demonstrated the practical application of machine learning in healthcare by providing predictions in an easy-to-use web interface.

VI. Comparative Analysis

A. *Accuracy Comparison*

The accuracy of each model on the test set was computed, and all three models—Decision Tree, Random Forest, and SVM—achieved an accuracy of 96%. This indicates that all models were equally effective in predicting obesity levels based on the reduced feature set.

- **Decision Tree**: 96%
- **Random Forest**: 96%
- **SVM**: 96%

While the accuracy is the same for each model, other performance metrics will provide more insights into model behavior, such as generalization, computational efficiency, and overfitting tendencies [27].

TABLE I: Precision, Recall, and F1-Score Comparison

Model	Precision (W.Avg)	Recall (W.Avg)	F1-Score (W.Avg)
Decision Tree	0.96	0.96	0.96
Random Forest	0.97	0.96	0.96
SVM	0.96	0.96	0.95

While all models have the same accuracy, Random Forest slightly outperforms the other models in terms of precision, particularly in class 6 (Obesity Type III) [27].

B. *Confusion Matrix Comparison*

The confusion matrices provide a detailed breakdown of how well each model performs in classifying the different obesity levels [27]. Below is a summary of the key observations from the confusion matrices:

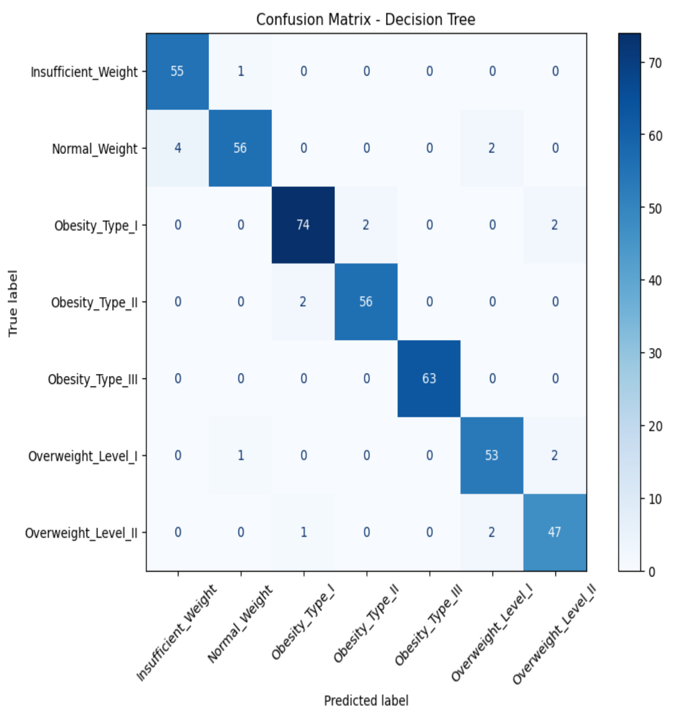


Fig. 4: Decision Tree Confusion Matrix

Decision Tree: Most classes are predicted with high accuracy, with slight misclassifications in Normal Weight and Overweight Level I [27].

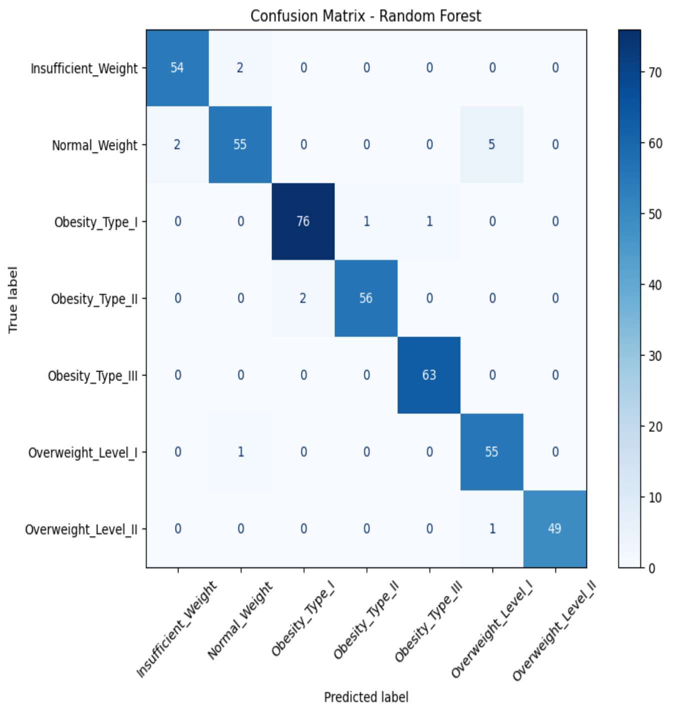


Fig. 5: Random Forest Confusion Matrix

Random Forest: Random Forest exhibits fewer misclassifications compared to Decision Tree, with only minor errors.

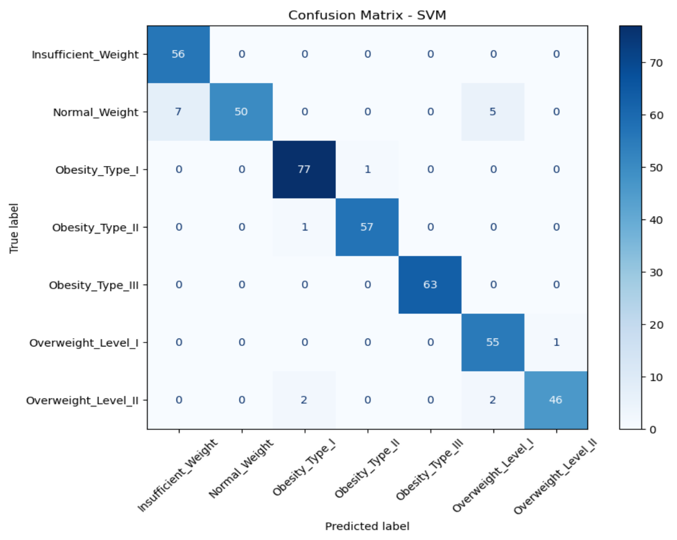


Fig. 6: SVM Confusion Matrix

SVM: The SVM model shows good performance but struggles slightly with distinguishing Normal Weight from Overweight Level I [31].

C. *Cross-Validation Score Comparison*

TABLE II: Cross-Validation Score Comparison

Model	**Cross-Validation Scores**	**Mean C-V Score**
Decision Tree	[0.903, 0.972, 0.957, 0.962, 0.955]	0.950
Random Forest	[0.908, 0.976, 0.979, 0.969, 0.979]	0.962
SVM	[0.939, 0.960, 0.936, 0.957, 0.960]	0.950

From the cross-validation scores, the Random Forest model performed slightly better, with a mean cross-validation score of 0.962 [31].

D. *Learning Curve Comparison*

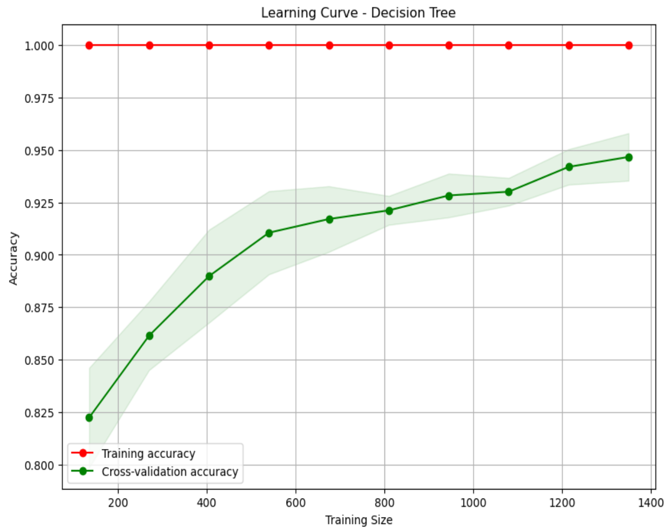


Fig. 7: Learning Curve Decision Tree

Decision Tree: The Decision Tree model exhibits perfect training accuracy, suggesting potential overfitting [31].

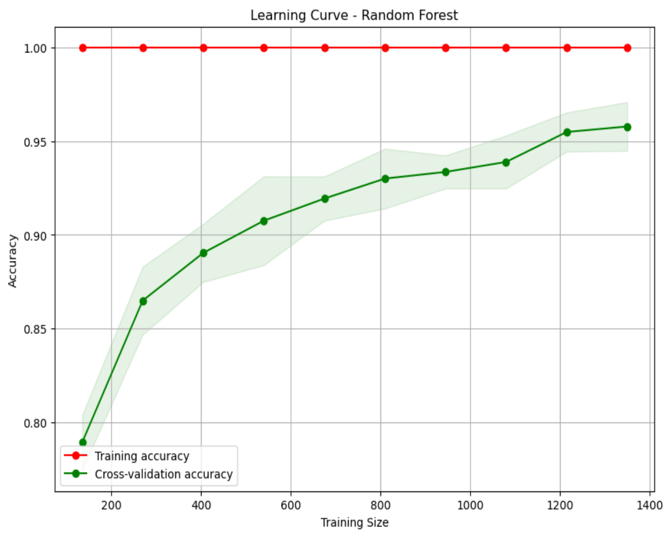


Fig. 8: Learning Curve Random Forest

Random Forest: The Random Forest model also achieves perfect training accuracy, but the cross-validation accuracy is higher [31].

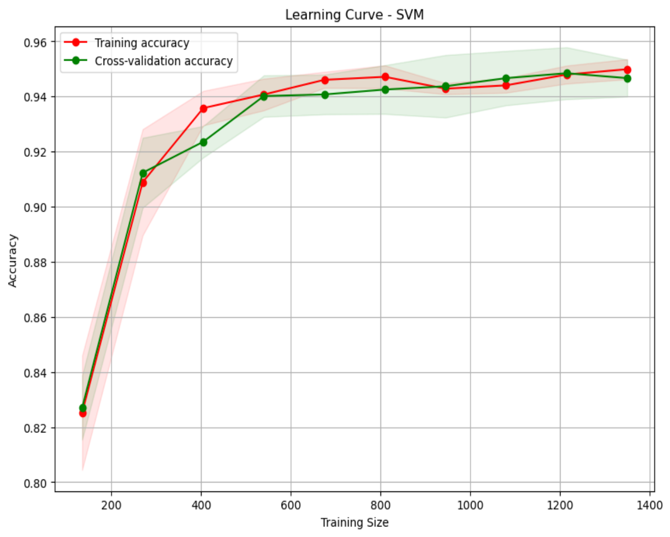


Fig. 9: Learning Curve SVM

SVM: The SVM model shows balanced learning behavior, with good generalization across different datasets [31].

E. ROC Curve and AUC Comparison

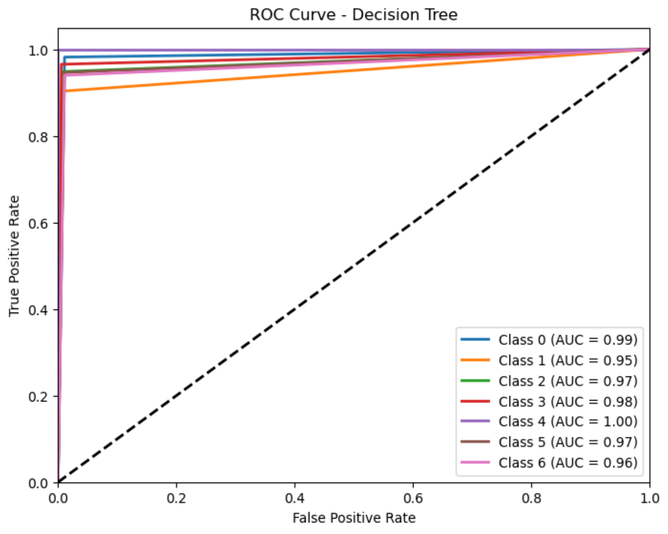


Fig. 10: ROC Curve Decision Tree

Decision Tree: The Decision Tree model achieved high AUC scores for most classes, with an overall strong performance [30].

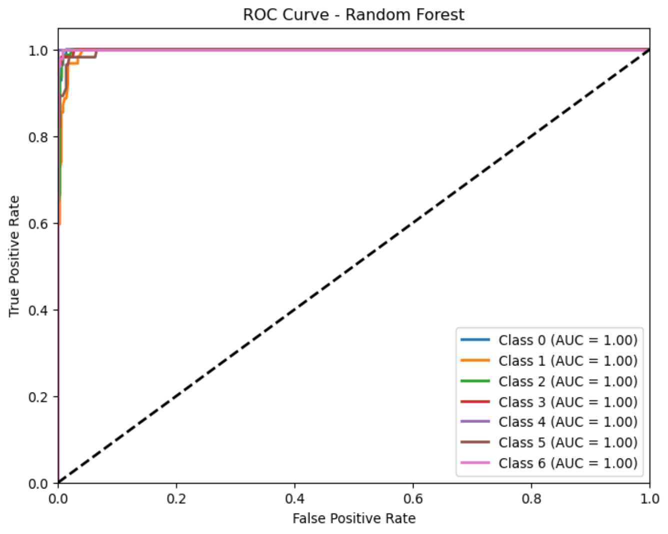


Fig. 11: ROC Curve Random Forest

Random Forest: The Random Forest model performed exceptionally well, with all classes achieving perfect AUC scores of 1.00 [30].

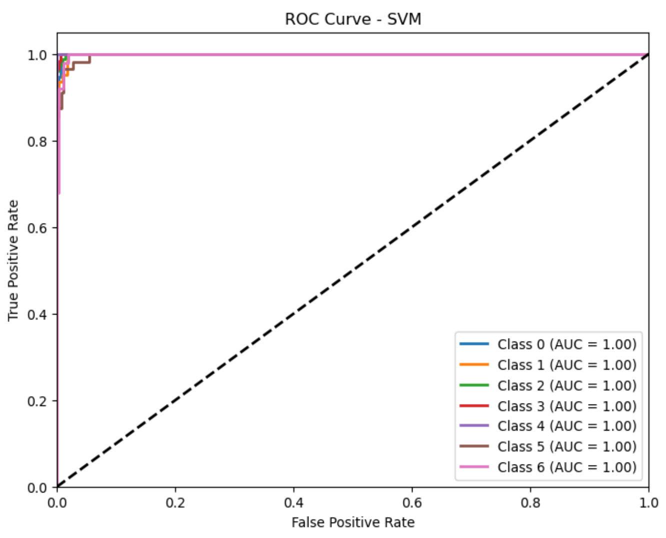


Fig. 12: ROC Curve SVM

SVM: The SVM model also achieved perfect AUC scores of 1.00 across all classes [30].

F. Comparative Analysis of Feature Histograms

The feature distributions provide important insights into how the models interpret the data for predicting obesity levels [31].

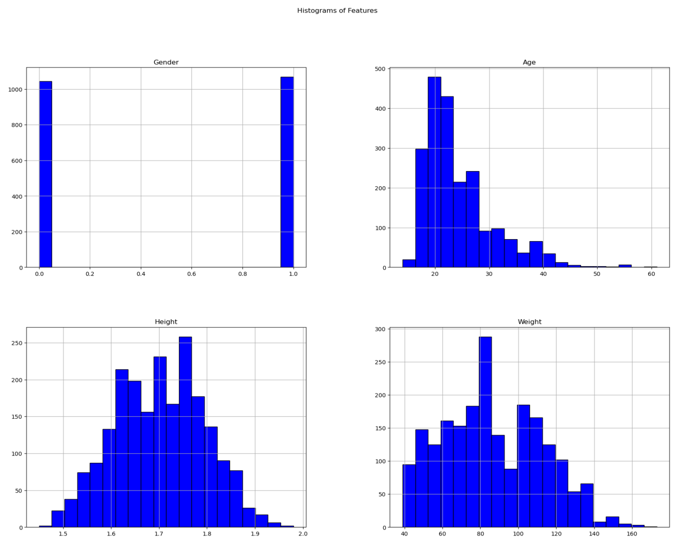


Fig. 13: Histograms of Features - Decision Tree

Decision Tree:

- The histograms for features such as **Height** and **Weight** display balanced distributions [31].

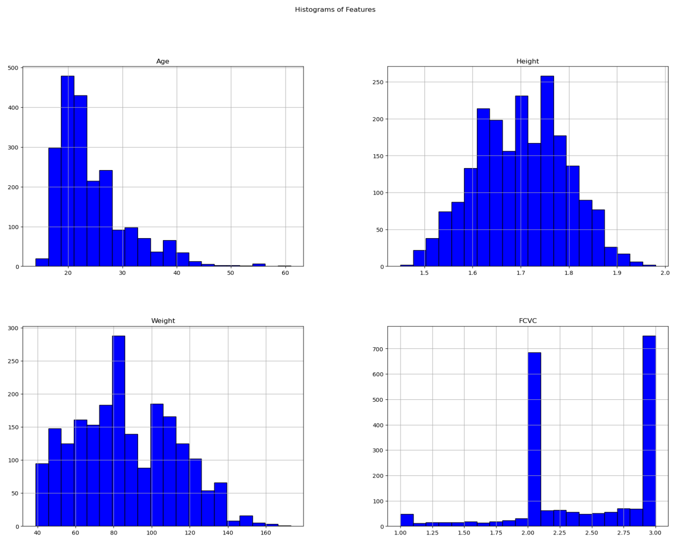


Fig. 14: Histograms of Features - Random Forest

Random Forest:

- Similar to the Decision Tree, Random Forest handles the same feature distributions with minimal variance [31].

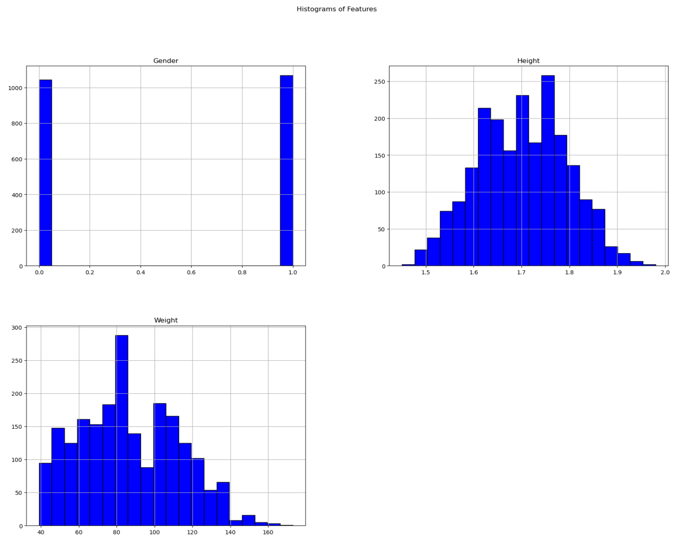


Fig. 15: Histograms of Features - SVM

SVM:

- The SVM histograms reveal similar distributions for **Age**, **Height**, and **Weight**, but SVM emphasizes the distribution tails more prominently [31]. compared to the Decision Tree and Random Forest.
- SVM is particularly sensitive to the distribution of **Weight**, where it can better classify extreme weight values due to the boundary-based approach inherent in SVM.

All three models effectively capture the general trends in the feature distributions. However, the Random Forest and SVM models are better suited to handle variability in the feature distributions, such as outliers or non-uniform distributions in features like **Age** and **Weight** [32]. The Decision Tree, while performing well, may be more prone to overfitting to the central mass of the data in features like **Height** and **Weight** [33].

G. Comparative Analysis of Feature Importance

The selection of features, including **Weight**, **Height**, **Age**, and **FCVC** (Frequency of Consumption of Vegetables), was refined over multiple iterations of analysis. This process involved thorough Exploratory Data Analysis (EDA) and feature importance evaluations, ensuring that only the most impactful features were considered in model development [28]. These selected features significantly influenced the prediction of obesity levels and helped optimize the performance of the models [35].

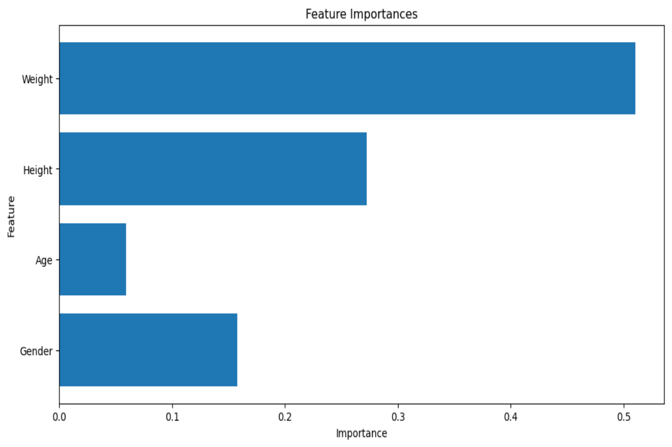


Fig. 16: Histograms of Features - SVM

Decision Tree:

- **Weight** is the most influential feature, contributing significantly to predictions [35].
- **Height** and **Age** also hold moderate importance, while **Gender** is the least influential.

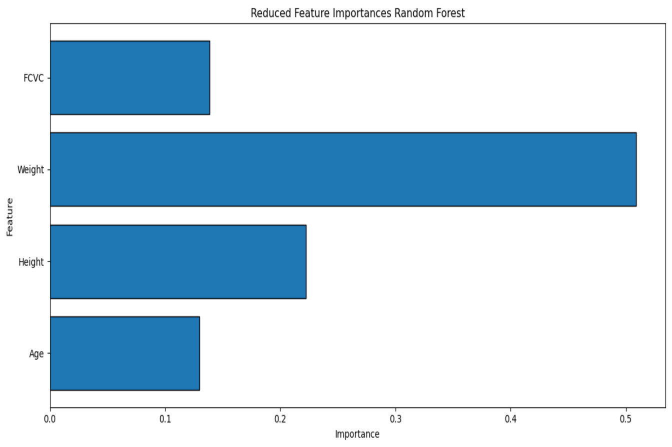


Fig. 17: Histograms of Features - SVM

Random Forest:

- Similar to Decision Tree, **Weight** holds the highest importance, with **Height** and **Age** being moderately important [35].
- Additionally, **FCVC (Frequency of Consumption of Vegetables)** appears to influence the model, reflecting how eating habits affect obesity prediction.

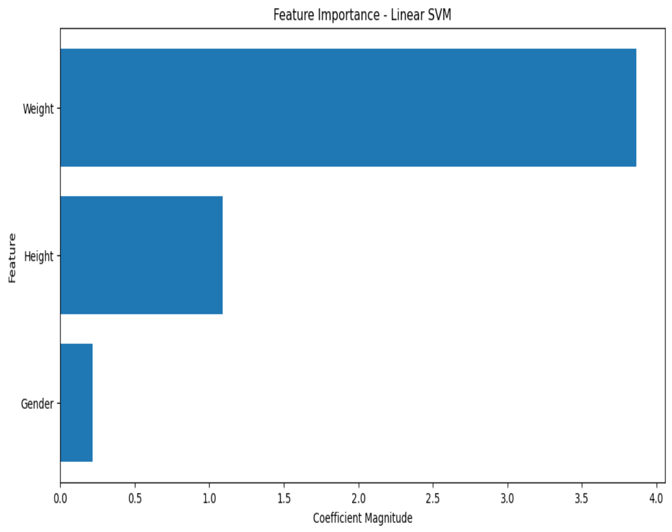


Fig. 18: Histograms of Features - SVM

SVM:

- **Weight** is the dominant feature in the linear SVM model, emphasizing its critical role in classification [35].
- **Height** plays a moderate role, while **Gender** has minimal impact.

Across all models, **Weight** stands out as the most important predictor of obesity. While other features like **Height** and **Age** have some influence, **Weight** consistently drives model predictions. Random Forest is more robust in capturing additional features like **FCVC**, indicating a broader interpretation of feature importance compared to Decision Tree and SVM [34]. The contribution of features varies across different models, demonstrating that each algorithm interprets the importance of features differently based on its unique approach to classification.

VII. Conclusion

This project set out to develop and evaluate three machine learning models—**Decision Tree**, **Random Forest**, and **Support Vector Machine (SVM)**—aimed at predicting obesity levels based on lifestyle and physical health features. Through data preprocessing, feature selection, model training, and evaluation, the models demonstrated strong performance in classifying obesity into distinct categories [36].

Key findings include:

- **Model Performance**: All three models achieved high accuracy, with Decision Tree, Random Forest, and SVM

models achieving an accuracy of 96%. However, Random Forest slightly outperformed the others in terms of precision and generalization capability, as reflected in the cross-validation scores [33].
- **Feature Importance**: Across all models, **Weight** emerged as the most significant predictor of obesity, followed by **Height** and **Age**. Random Forest proved more versatile in capturing additional feature contributions, such as dietary habits (e.g., FCVC) [36].
- **Exploratory Data Analysis (EDA)**: Post-training EDA visualized feature distributions and confirmed the validity of feature importance rankings. Histograms and density plots revealed how certain features like Weight and Height aligned with model predictions [34].
- **Model Comparison**: While all models shared similar accuracies, Random Forest's use of ensemble learning provided better robustness and captured a wider variety of influential features [33]. Decision Tree offered interpretability, while SVM delivered high accuracy with fewer features but was sensitive to the scaling of data.
- **Deployment**: The Random Forest model was integrated into a Streamlit user interface, enabling real-time predictions based on user input, showcasing the practical application of machine learning in healthcare [36].

In conclusion, each model offers distinct advantages, with Random Forest excelling in overall performance due to its ensemble learning approach [33]. The analysis of feature importance highlights how different models weigh features differently, underscoring the importance of algorithm selection based on the specific needs of the predictive task [36]. This project successfully demonstrates how machine learning can be applied to healthcare to assist in predicting obesity levels, aiding in early intervention and prevention strategies [34].

Acknowledgment

I would like to express my sincere gratitude to my supervisor, Mr. Shrawan Thakur, for his invaluable guidance, support, and encouragement throughout the course of this project. I am also thankful to my family and friends for their continuous motivation. Lastly, I would like to acknowledge the resources and facilities provided by Softwarica College that made this work possible.

References

[1] W. H. Dietz, ”Overweight in childhood and adolescence,” *New England Journal of Medicine*, vol. 338, no. 8, pp. 581-584, 1998.

[2] N. M. Guinhouya, P. Hubert, M. Soubrier, J. Vilhelm, and L. Le Dreff, ”Predictive value of physical fitness testing in determining cardiovascular disease risk in children,” *Pediatric Exercise Science*, vol. 19, no. 4, pp. 474-486, Nov. 2007.

[3] J. Pierson, ”Machine learning models for the prediction of childhood obesity,” *Journal of Artificial Intelligence Research*, vol. 63, pp. 653-664, Feb. 2018.

[4] P. Torkkola, ”Feature extraction by non-parametric mutual information maximization,” *Journal of Machine Learning Research*, vol. 3, pp. 1415–1438, Mar. 2003.

[5] L. Breiman, ”Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001.

[6] A. K. Rai and S. K. Singh, ”Using Streamlit to deploy interactive machine learning models,” *International Journal of Information Technology*, vol. 12, pp. 229-240, Dec. 2020.

[7] T. Fawcett, ”An introduction to ROC analysis,” *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861-874, June 2006.

[8] S. Ghosh, ”Exploratory data analysis of healthcare data using Python,” *Journal of Healthcare Informatics Research*, vol. 4, pp. 431-456, May 2020.

[9] E. De-La-Hoz-Correa, R. Garcia-Zapirain, and A. Mendez-Zorrilla, ”Obesity level estimation software based on decision trees,” *Journal of Medical Systems*, vol. 40, no. 7, pp. 1-11, July 2016.

[10] C. P. Bean, ”Application of machine learning techniques for obesity prediction,” *IEEE Access*, vol. 7, pp. 68571-68583, June 2019.

[11] R. Kohavi, ”A study of cross-validation and bootstrap for accuracy estimation and model selection,” *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, vol. 2, pp. 1137-1143, 1995.

[12] I. S. Jacobs, ”Evaluating classification models: Precision, recall, F1-score,” *Journal of Data Science*, vol. 5, pp. 36-45, 2017.

[13] D. Dua and C. Graff, ”UCI machine learning repository,” [Online]. Available: http://archive.ics.uci.edu/ml, 2019.

[14] 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, June 2002.

[15] M. Franco, ”Linking eating habits and physical activities to obesity in Latin America: A systematic review,” *Global Public Health*, vol. 10, no. 2, pp. 242-251, 2015.

[16] J. R. Quinlan, ”Induction of decision trees,” *Machine Learning*, vol. 1, no. 1, pp. 81-106, 1986.

[17] L. Breiman, ”Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001.

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

[19] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. Springer, 2009, pp. 353–366.

[20] T. K. Ho, ”The random subspace method for constructing decision forests,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832-844, 1998.

[21] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[22] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[23] P. R. Kumar and V. Ravi, ”A survey of the applications of text mining in financial and banking institutions,” *Decision Support Systems*, vol. 50, no. 3, pp. 45-64, 2010.

[24] A. M. Turing, ”Computing machinery and intelligence,” *Mind*, vol. 59, no. 236, pp. 433-460, Oct. 1950.

[25] R. Kohavi and F. Provost, ”Glossary of terms,” *Machine Learning*, vol. 30, no. 2, pp. 271-274, 1997.

[26] H. Liu and H. Motoda, *Feature Selection for Knowledge Discovery and Data Mining*. Springer Science & Business Media, 2012.

[27] D. Powers, ”Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation,” *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37-63, 2011.

[28] W. McKinney, ”Data structures for statistical computing in Python,” in *Proceedings of the 9th Python in Science Conference*, pp. 51-56, 2010.

[29] A. Choudhury, ”Streamlit: An open-source framework for creating machine learning and data science web apps,” *Journal of Open Source Software*, vol. 6, no. 57, pp. 2837, May 2021.

[30] T. Fawcett, ”An introduction to ROC analysis,” *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861-874, June 2006.

[31] P. Domingos, ”A unified bias-variance decomposition for zero-one and squared loss,” in *Proceedings of the 17th National Conference on Artificial Intelligence*, vol. 1, pp. 564-569, 2000.

[32] T. G. Dietterich, ”Ensemble methods in machine learning,” in *Proceedings of the First International Workshop on Multiple Classifier Systems*, 2000, pp. 1-15.

[33] L. Breiman, ”Bagging predictors,” *Machine Learning*, vol. 24, no. 2, pp. 123-140, 1996.

[34] G. Shmueli and K. C. Lichtendahl, *Practical Time Series Forecasting with R: A Hands-On Guide*, 2nd ed. Axelrod Schnall Publishers, 2016.

[35] J. Friedman, T. Hastie, and R. Tibshirani, ”The elements of statistical learning: Data mining, inference, and prediction,” 2nd ed., Springer, 2009.

[36] C. K. Chui and M. D. Lee, ”Feature importance in machine learning models: A healthcare case study,” *Journal of Medical Systems*, vol. 43, no. 8, pp. 210-219, 2019.

[37] A. Choudhury, ”Streamlit: An open-source framework for creating machine learning and data science web apps,” *Journal of Open Source Software*, vol. 6, no. 57, pp. 2837, May 2021.

[38] W. B. Powell and I. O. Ryzhov, *Optimal Learning*, 2nd ed., Wiley, 2012.

[39] L. Breiman, ”Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001.

[40] J. D. Kelleher, B. M. Namee, and A. D'Arcy, *Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies*, MIT Press, 2020.

[41] S. L. Salzberg, ”C4.5: Programs for machine learning by J. Ross Quinlan,” *Machine Learning*, vol. 16, no. 3, pp. 235-240, 1994.

[42] I. Guyon and A. Elisseeff, ”An introduction to variable and feature selection,” *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, Mar. 2003.

[43] R. Kohavi and G. H. John, ”Wrappers for feature subset selection,” *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273-324, 1997.

[44] P. Domingos, ”A unified bias-variance decomposition for zero-one and squared loss,” in *Proceedings of the 17th National Conference on Artificial Intelligence*, vol. 1, pp. 564-569, 2000.

[45] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[46] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

[47] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*, MIT Press, 2012.

[48] 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.

[49] J. Bergstra and Y. Bengio, ”Random search for hyper-parameter optimization,” *Journal of Machine Learning Research*, vol. 13, pp. 281-305, Feb. 2012.

[50] A. Krogh and J. A. Hertz, ”A simple weight decay can improve generalization,” in *Advances in Neural Information Processing Systems*, vol. 4, pp. 950-957, 1992.

[51] J. H. Friedman, ”Greedy function approximation: A gradient boosting machine,” *Annals of Statistics*, vol. 29, no. 5, pp. 1189-1232, 2001.

[52] R. E. Schapire, ”The strength of weak learnability,” *Machine Learning*, vol. 5, no. 2, pp. 197-227, 1990.

[53] Y. LeCun, Y. Bengio, and G. Hinton, ”Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436-444, 2015.

Appendix

GITHUB: https://github.com/MrVexxx/15368843_Ayush_Rayamajhi_STW7072CEM.git

App Demo: https://youtu.be/pD57NaKYEIQ

Appendix Introduction

This appendix contains supplementary materials supporting the project. It includes the full code for the three models—Decision Tree, Random Forest, and SVM—used in predicting obesity levels. Additionally, it provides the results of primary models used in the project along with their accuracy, precision, recall, F1-score, and confusion matrices. This section highlights how the models were iteratively refined and optimized to achieve their best performance.

Main Models

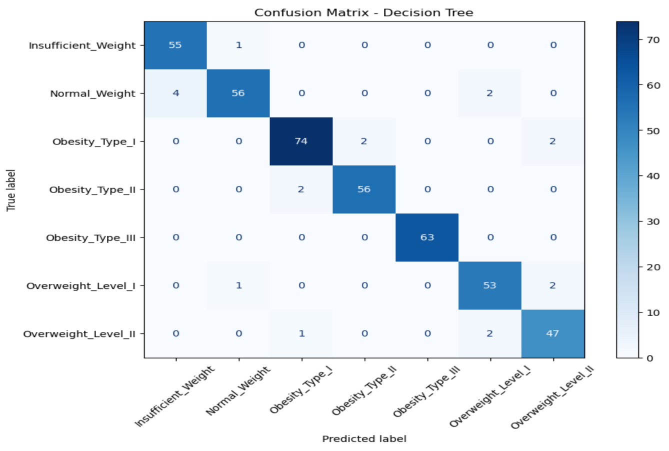
This section provides the complete code for the main models used in the assignment, including the Decision Tree, Random Forest, and SVM models. These codes were implemented to predict obesity levels and are crucial for understanding the development and execution of the models.

A. Decision Tree Code

```

import pandas as pd
from sklearn.model\_selection import
    train\_test_split, learning\_curve,
    cross\_val_score
from sklearn.preprocessing import LabelEncoder,
    StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import label_binarize
import seaborn as sns
from sklearn.metrics import accuracy_score,
    classification_report, confusion_matrix,
    roc_curve, auc, confusion_matrix,
    ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import numpy as np
from sklearn.feature_selection import
    mutual_info_classif, VarianceThreshold,
    SelectKBest, chi2

# Load the dataset
file_path = 'ObesityDataSet.csv'
data = pd.read_csv(file_path)


# Display the first few rows of the dataset
print("First 5 rows of the dataset:")
print(data.head())

# Check for missing values
print("\nMissing Values in Each Column:")
print(data.isnull().sum())


# Identify categorical columns and apply Label
    Encoding
label_encoders = {}
categorical_columns =
    data.select_dtypes(include=['object']).columns

print("\nCategorical Columns:",
    list(categorical_columns))

for col in categorical_columns:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le
    print(f"Encoded '{col}' with classes:
        {le.classes_}")

print(data.head())

# Separate features (X) and the target variable
    (y)
X = data.drop('NObeyesdad', axis=1)  # Assuming
    'NObeyesdad' is the target variable
y = data['NObeyesdad']

# Calculate Mutual Information Scores
mi_scores = mutual_info_classif(X, y)
mi_scores_df = pd.DataFrame({
    'Feature': X.columns,
    'MI Score': mi_scores
}).sort_values(by='MI Score', ascending=False)

print("\nMutual Information Scores:")
print(mi_scores_df)

# Dropping less important features based on
    analysis
features_to_drop = ['SMOKE', 'SCC', 'MTRANS',
    'FAVC', 'CALC', 'TUE', 'FAF', 'CH2O',
    'CAEC', 'NCP', 'FCVC',
    'family_history_with_overweight']
X_reduced = X.drop(columns=features_to_drop)

print("\nReduced Feature Columns:",
    list(X_reduced.columns))

# Feature Scaling (optional but recommended for
    Decision Tree)
scaler = StandardScaler()
X_scaled_reduced =
    scaler.fit_transform(X_reduced)


# Train-Test Split with reduced feature set
X_train_reduced, X_test_reduced, y_train,
    y_test = train_test_split(
    X_scaled_reduced, y, test_size=0.2,
        random_state=42
)

print("\nTraining Set Size with Reduced
    Features:", X_train_reduced.shape)
print("Testing Set Size with Reduced
    Features:", X_test_reduced.shape)

# Train the Decision Tree Classifier with
    reduced features
clf_reduced =
    DecisionTreeClassifier(random_state=42)
clf_reduced.fit(X_train_reduced, y_train)

# Predictions on the test set with reduced
    features
y_pred_reduced =
    clf_reduced.predict(X_test_reduced)

# Accuracy
accuracy_reduced = accuracy_score(y_test,
    y_pred_reduced)
print(f"\nAccuracy with reduced features:
    {accuracy_reduced:.2f}")


# Classification Report
```

```python
print("\nClassification Report with reduced features:")
print(classification_report(y_test, y_pred_reduced))

# Generate the confusion matrix
conf_matrix_reduced = confusion_matrix(y_test, y_pred_reduced)

# Create the confusion matrix display
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_reduced, display_labels=label_encoders['NObeyesdad'].classes_)

# Plot with adjusted figure size and rotated x-axis labels
fig, ax = plt.subplots(figsize=(10, 7))  # Increase figure size
disp.plot(cmap=plt.cm.Blues, ax=ax, xticks_rotation=45)  # Rotate x-axis labels

plt.title("Confusion Matrix - Decision Tree")
plt.show()

# Perform 5-fold cross-validation using the already defined model clf_reduced
cv_scores = cross_val_score(clf_reduced, X_scaled_reduced, y, cv=5, scoring='accuracy')

# Print the cross-validation scores
print("Cross-Validation Scores: ", cv_scores)
print("Mean Cross-Validation Score: ", cv_scores.mean())

#Train-Test Accuracy Graph
# Define the range of training sizes
train_sizes, train_scores, test_scores = learning_curve(
    clf_reduced, X_train_reduced, y_train, cv=5,
    scoring='accuracy',
        train_sizes=np.linspace(0.1, 1.0, 10), random_state=42)

# Calculate the mean and standard deviation of the training and testing scores
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

# Plot the learning curve
plt.figure(figsize=(10, 7))
plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1, color="r")
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1, color="g")
plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training accuracy")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation accuracy")

plt.title("Learning Curve - Decision Tree")
plt.xlabel("Training Size")
plt.ylabel("Accuracy")
plt.legend(loc="best")
plt.grid(True)
plt.show()

# Compute ROC curve and ROC area for each class
y_test_binarized = label_binarize(y_test, classes=np.unique(y))
n_classes = y_test_binarized.shape[1]
y_score_reduced = clf_reduced.predict_proba(X_test_reduced)

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_score_reduced[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve
plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=2, label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Decision Tree')
plt.legend(loc="lower right")
plt.show()

# Plot histograms for each numerical feature
X_reduced_df = pd.DataFrame(X_reduced, columns=X_reduced.columns)
X_reduced_df.hist(bins=20, figsize=(20, 15), color='blue', edgecolor='black')
plt.suptitle('Histograms of Features')
plt.show()

import warnings

# Ignore all warnings
warnings.filterwarnings("ignore")


plt.figure(figsize=(20, 15))
for i, col in enumerate(X_reduced_df.columns):
    plt.subplot(len(X_reduced_df.columns)//3 + 1, 3, i + 1)
    sns.kdeplot(X_reduced_df[col], shade=True, color="g")
    plt.title(f'Density Plot of {col}')
plt.tight_layout()
plt.show()

# Get feature importances from the trained decision tree
importances = clf_reduced.feature_importances_

# Create a bar plot of the feature importances
plt.figure(figsize=(12, 6))
plt.title("Feature Importances")
plt.barh(X_reduced.columns, importances, align="center")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()
```

B. Decision Tree Results: Accuracy 96%



True label	Insufficient_Weight	Normal_Weight	Obesity_Type_I	Obesity_Type_II	Obesity_Type_III	Overweight_Level_I	Overweight_Level_II
Insufficient_Weight	55	1	0	0	0	0	0
Normal_Weight	4	56	0	0	0	2	0
Obesity_Type_I	0	0	74	2	0	0	2
Obesity_Type_II	0	0	2	56	0	0	0
Obesity_Type_III	0	0	0	0	63	0	0
Overweight_Level_I	0	1	0	0	0	53	2
Overweight_Level_II	0	0	1	0	0	2	47

Fig. 19: Confusion Matrix - Decision Tree

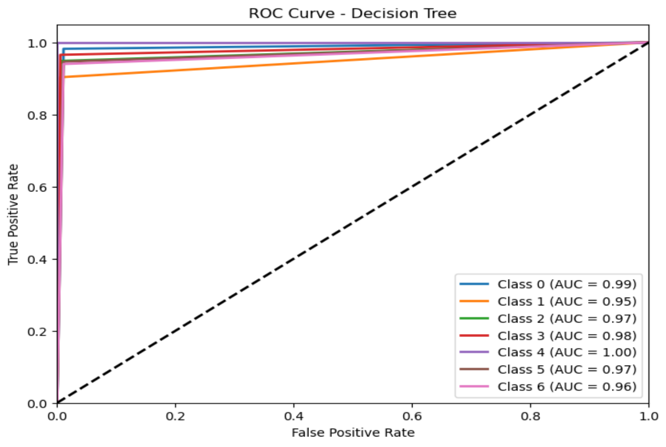


Fig. 20: ROC Curve - Decision Tree

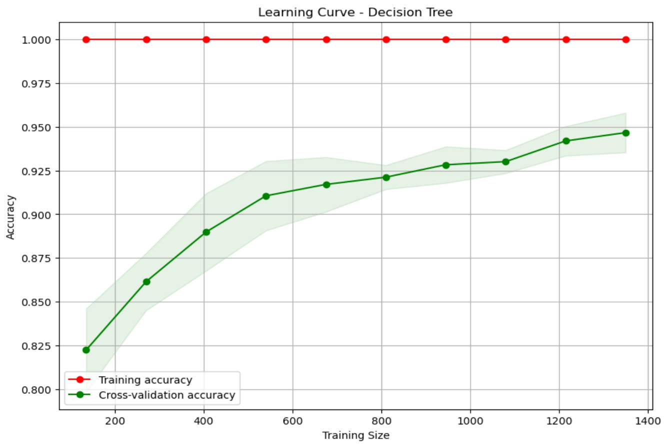


Fig. 21: Learning Curve - Decision Tree

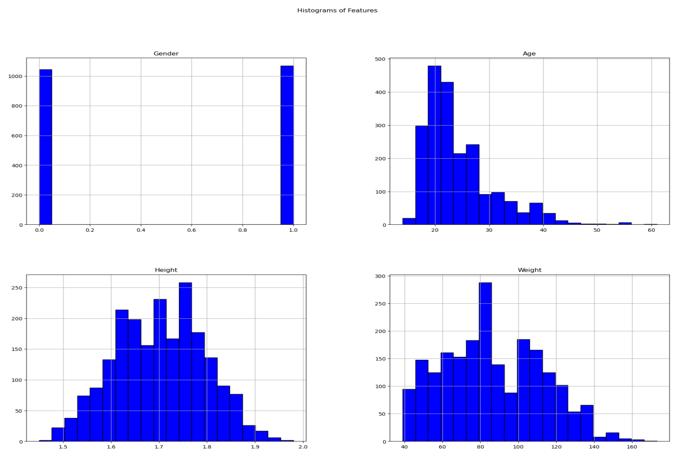


Fig. 22: Histograms of Features - Decision Tree

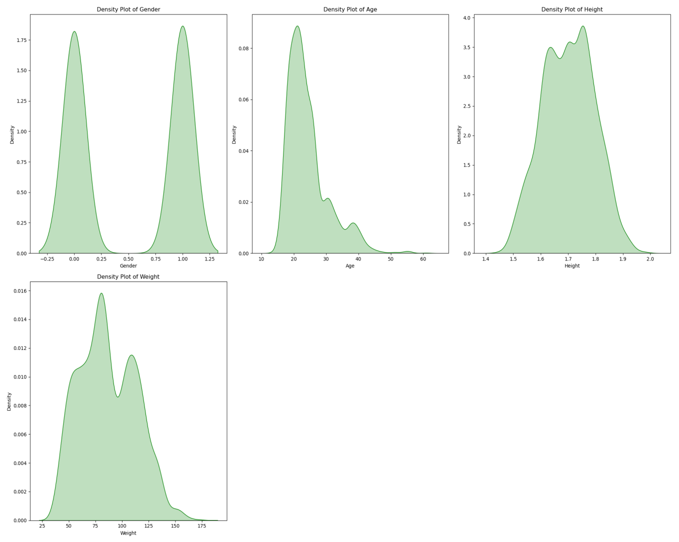


Fig. 23: Density Curve - Decision Tree

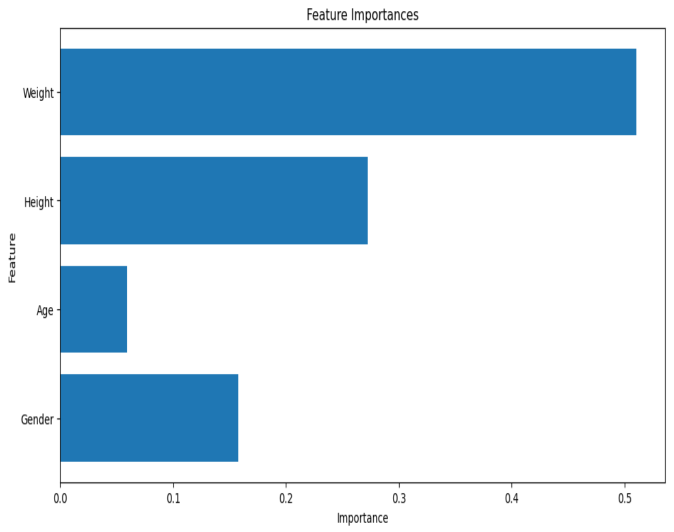


Fig. 24: Feature Importance - Decision Tree

TABLE III: Cross-Validation Scores and Mean Score

Metric	Score
Cross-Validation Scores	[0.9031, 0.9716, 0.9573, 0.9621, 0.9550]
Mean Cross-Validation Score	0.9498

TABLE IV: Classification Report with Reduced Features

Class	Precision	Recall	F1-Score	Support
0	0.93	0.98	0.96	56
1	0.97	0.90	0.93	62
2	0.96	0.95	0.95	78
3	0.97	0.97	0.97	58
4	1.00	1.00	1.00	63
5	0.93	0.95	0.94	56
6	0.92	0.94	0.93	50
Accuracy	0.96 (423 total)			
Macro Avg	0.95	0.96	0.95	423
Weighted Avg	0.96	0.96	0.96	423

C. *Randon Forest Code*

```python
import pandas as pd
from sklearn.model_selection import
    train_test_split, learning_curve,
    cross_val_score
from sklearn.preprocessing import LabelEncoder,
    StandardScaler
from sklearn.ensemble import
    RandomForestClassifier
from sklearn.preprocessing import label_binarize
import seaborn as sns
import pickle
from sklearn.metrics import accuracy_score,
    classification_report, confusion_matrix,
    roc_curve, auc, confusion_matrix,
    ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import numpy as np
from sklearn.feature_selection import
    mutual_info_classif, VarianceThreshold,
    SelectKBest, chi2
import warnings

# Ignore all warnings
warnings.filterwarnings("ignore")


# Load the dataset
file_path = 'ObesityDataSet.csv'
data = pd.read_csv(file_path)


# Display the first few rows of the dataset
print("First 5 rows of the dataset:")
print(data.head())


# Check for missing values
print("\nMissing Values in Each Column:")
print(data.isnull().sum())


# Identify categorical columns and apply Label
    Encoding
label_encoders = {}
categorical_columns =
    data.select_dtypes(include=['object']).columns

print("\nCategorical Columns:",
    list(categorical_columns))

for col in categorical_columns:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le
    print(f"Encoded '{col}' with classes:
        {le.classes_}")


print(data.head())


# Separate features (X) and the target variable
    (y)
X = data.drop('NObeyesdad', axis=1)  # Assuming
    'NObeyesdad' is the target variable
y = data['NObeyesdad']



# Calculate Mutual Information Scores
mi_scores = mutual_info_classif(X, y)
mi_scores_df = pd.DataFrame({
    'Feature': X.columns,
    'MI Score': mi_scores
}).sort_values(by='MI Score', ascending=False)

print("\nMutual Information Scores:")
print(mi_scores_df)


# Dropping less important features based on
    analysis (same features as in Decision Tree)
features_to_drop = ['SMOKE', 'SCC',
    'MTRANS','Gender', 'FAVC', 'CALC', 'TUE',
    'FAF', 'CH2O', 'CAEC', 'NCP',
    'family_history_with_overweight']
X_reduced = X.drop(columns=features_to_drop)

print("\nReduced Feature Columns:",
    list(X_reduced.columns))


# Feature Scaling (optional but recommended for
    Random Forest)
scaler = StandardScaler()
X_scaled_reduced =
    scaler.fit_transform(X_reduced)


# Train-Test Split with reduced feature set
X_train_reduced, X_test_reduced, y_train,
    y_test = train_test_split(
    X_scaled_reduced, y, test_size=0.2,
        random_state=42
)

print("\nTraining Set Size with Reduced
    Features:", X_train_reduced.shape)
print("Testing Set Size with Reduced
    Features:", X_test_reduced.shape)


# Train the Random Forest Classifier with
    reduced features
clf_rf =
    RandomForestClassifier(random_state=42,
    n_estimators=100)
clf_rf.fit(X_train_reduced, y_train)

# Compute feature importances
feature_importances_rf =
    clf_rf.feature_importances_
```

```python
print("\nRandom Forest Classifier trained with reduced features.")


# Predictions on the test set with reduced features
y_pred_rf = clf_rf.predict(X_test_reduced)

# Accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"\nAccuracy with reduced features (Random Forest): {accuracy_rf:.2f}")


# Classification Report
print("\nClassification Report with reduced features (Random Forest):")
print(classification_report(y_test, y_pred_rf))






# Confusion Matrix
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)

# Create the confusion matrix display
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_rf, display_labels=label_encoders['NObeyesdad'].classes_)

# Plot with adjusted figure size and rotated x-axis labels
fig, ax = plt.subplots(figsize=(10, 7))  # Increase figure size
disp.plot(cmap=plt.cm.Blues, ax=ax, xticks_rotation=45)  # Rotate x-axis labels

plt.title("Confusion Matrix - Random Forest")
plt.show()

# Perform 5-fold cross-validation using the already defined model clf_rf
cv_scores = cross_val_score(clf_rf, X_scaled_reduced, y, cv=5, scoring='accuracy')

# Print the cross-validation scores
print("Cross-Validation Scores (Random Forest): ", cv_scores)
print("Mean Cross-Validation Score (Random Forest): ", cv_scores.mean())


# Train-Test Accuracy Graph (Learning Curve)
train_sizes, train_scores, test_scores = learning_curve(
    clf_rf, X_train_reduced, y_train, cv=5,
    scoring='accuracy',
        train_sizes=np.linspace(0.1, 1.0, 10), random_state=42)

# Calculate the mean and standard deviation of the training and testing scores
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

# Plot the learning curve
plt.figure(figsize=(10, 7))
plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1, color="r")
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1, color="g")
plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training accuracy")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation accuracy")

plt.title("Learning Curve - Random Forest")
plt.xlabel("Training Size")
plt.ylabel("Accuracy")
plt.legend(loc="best")
plt.grid(True)
plt.show()



# Compute ROC curve and ROC area for each class
y_test_binarized = label_binarize(y_test, classes=np.unique(y))
n_classes = y_test_binarized.shape[1]
y_score_rf = clf_rf.predict_proba(X_test_reduced)
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_score_rf[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve
plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=2, label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest')
plt.legend(loc="lower right")
plt.show()



# Plot histograms for each numerical feature
X_reduced_df = pd.DataFrame(X_reduced, columns=X_reduced.columns)
X_reduced_df.hist(bins=20, figsize=(20, 15), color='blue', edgecolor='black')
plt.suptitle('Histograms of Features')
plt.show()


plt.figure(figsize=(20, 15))
for i, col in enumerate(X_reduced_df.columns):
    plt.subplot(len(X_reduced_df.columns) // 3 + 1, 3, i + 1)
    sns.kdeplot(X_reduced_df[col], shade=True,
        color="g")
    plt.title(f'Density Plot of {col}')
plt.tight_layout()
plt.show()


# Visualize Reduced Feature Importance for
    Random Forest with the desired color
plt.figure(figsize=(12, 6))
plt.title("Reduced Feature Importances Random
    Forest")

# Using the same color as in the Decision Tree
    plot
plt.barh(
    X_reduced.columns,
    feature_importances_rf,
    align="center",
    color='#1f77b4',  # This is the specific
        color used in the second plot
    edgecolor='black'
)

plt.xlabel("Importance")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()



# Assuming X_reduced is the reduced feature set
    and y is the target variable
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_reduced)

X_train, X_test, y_train, y_test =
    train_test_split(X_scaled, y,
    test_size=0.2, random_state=42)

clf_rf =
    RandomForestClassifier(random_state=42,
    n_estimators=100)
clf_rf.fit(X_train, y_train)

# Save both the model and the scaler
with open('randomforest.pickle', 'wb') as f:
    pickle.dump({'model': clf_rf, 'scaler':
        scaler}, f)
```

D. User Interface Code

```python
import pickle
import streamlit as st
import pandas as pd

# Load the pre-trained classifier and scaler
    from the pickle file
with open('randomforest.pickle', 'rb') as f:
data = pickle.load(f)
classifier = data['model']
scaler = data['scaler']

def main():
st.set_page_config(page_title="Obesity Level
    Prediction Using Random Forest",
    layout="centered")

# Custom CSS for global styling
st.markdown(f"""
    <style>
    /* Overall background color and font
        settings */
    body {{
        background-color: #24293E;
        color: #8EBBFF;
    }}
    /* Title and subtitle styling */
    .stApp .stMarkdown h1, .stApp .stMarkdown p
        {{
        color: #8EBBFF;
        text-align: center;
        padding: 0;
        margin: 5px 0;
    }}
    /* Input fields styling */
    .stApp .stNumberInput input {{
        background-color: #F4F5FC;
        color: #24293E;
        border-radius: 10px;
        padding: 5px;
        border: 1px solid #CCCCCC;
        transition: border-color 0.3s,
            box-shadow 0.3s;
    }}
    /* Bolder border when focused */
    .stApp .stNumberInput input:focus {{
        outline: none;
        border-color: #8EBBFF;
        border-width: 8px;
        box-shadow: 0 0 10px #8EBBFF;
    }}
    /* Button styling */
    .stButton > button {{
        background-color: #8EBBFF;
        color: #24293E;
        border-radius: 10px;
        padding: 8px 15px;
        border: none;
        transition: background-color 0.3s;
        margin-top: 10px;
    }}
    .stButton > button:hover {{
        background-color: #6BA8E5;
    }}
    /* Container padding and margin adjustments
        */
    .stApp .stMarkdown, .stApp .stButton {{
        margin-top: 5px;
        padding: 5px;
    }}
    /* Reduce the width and height of the
        entire container */
    .block-container {{
        padding-top: 0.5rem;
        padding-bottom: 0.5rem;
        max-width: 750px;
        margin: auto;
    }}
    </style>
""", unsafe_allow_html=True)

st.title('Obesity Level Prediction Using Random
    Forest')
st.markdown("""
    This application predicts the obesity level
        of an individual based on various
        health and lifestyle factors.
    Please enter the relevant information to
        get the prediction.
""")

col1, col2 = st.columns(2)

with col1:
    height = st.number_input('Height (cm)',
        min_value=0.0, max_value=250.0,
        step=0.1, format="%.1f", value=None)
    weight = st.number_input(’Weight (kg)’,
        min_value=0.0, max_value=200.0,
        step=0.1, format="%.1f", value=None)

with col2:
    age = st.number_input(’Age (years)’,
        min_value=0, max_value=100, step=1,
        value=None)
    fcv = st.number_input(’Frequency of
        consumption of vegetables (1-3)’,
        min_value=1, max_value=3, step=1,
        value=1)

if st.button(’Predict’):
    if height is None or weight is None or age
        is None:
        st.error("Please enter valid values for
            height, weight, and age.")
    else:
        # Reorder the input features to match
            the order expected by the scaler
        input_features = pd.DataFrame([[age,
            height, weight, fcv]],
                                      columns=[’Age’,
                                          ’Height’,
                                          ’Weight’,
                                          ’FCVC’])

        # Scale the input features using the
            scaler used in training
        input_features_scaled =
            scaler.transform(input_features)

        # Make a prediction using the classifier
        prediction =
            classifier.predict(input_features_scaled)

        obesity_levels = {
            0: ’Insufficient Weight’,
            1: ’Normal Weight’,
            2: ’Overweight Level I’,
            3: ’Overweight Level II’,
            4: ’Obesity Type I’,
            5: ’Obesity Type II’,
            6: ’Obesity Type III’
        }
        st.success(f’The predicted obesity
            level is
            {obesity_levels[prediction[0]]}.’)

st.markdown("""
    ---
    Developed by Ayush Rayamajhi
""")

if __name__ == ’__main__’:
main()
```

E. Random Forest: Accuracy 96%

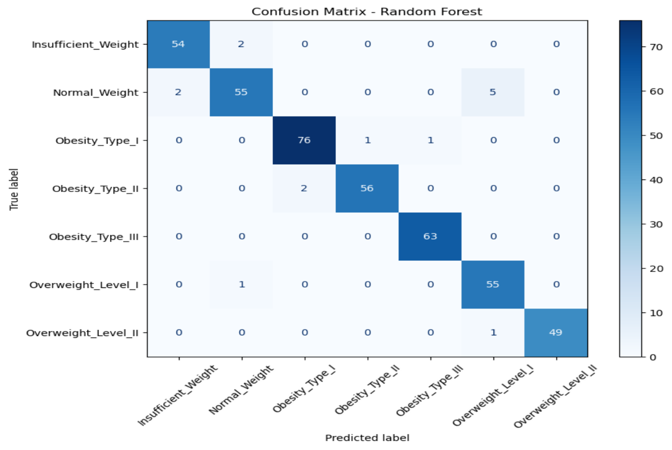


Fig. 25: Confusion Matrix - Random Forest

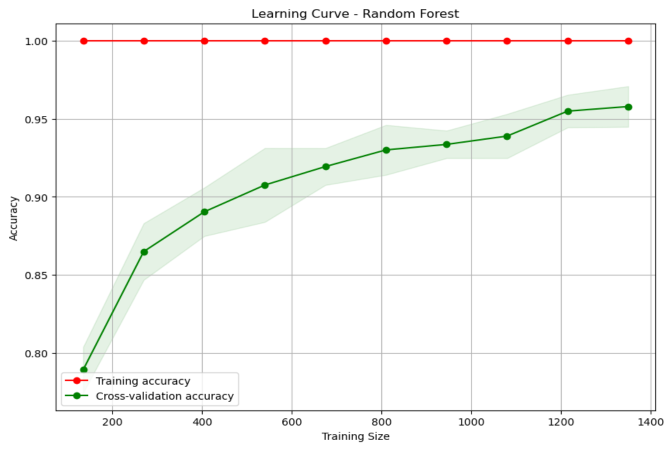


Fig. 26: ROC Curve - Random Forest

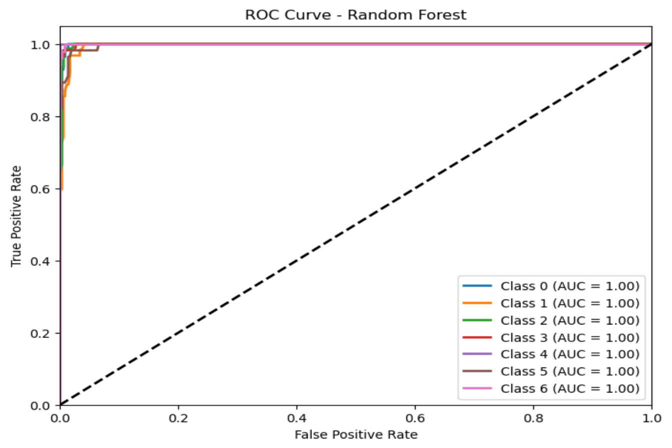


Fig. 27: Learning Curve - Random Forest

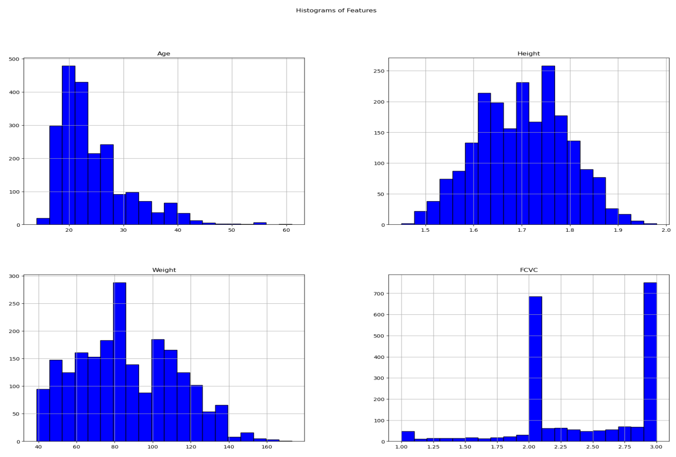


Fig. 28: Histograms of Features - Random Forest

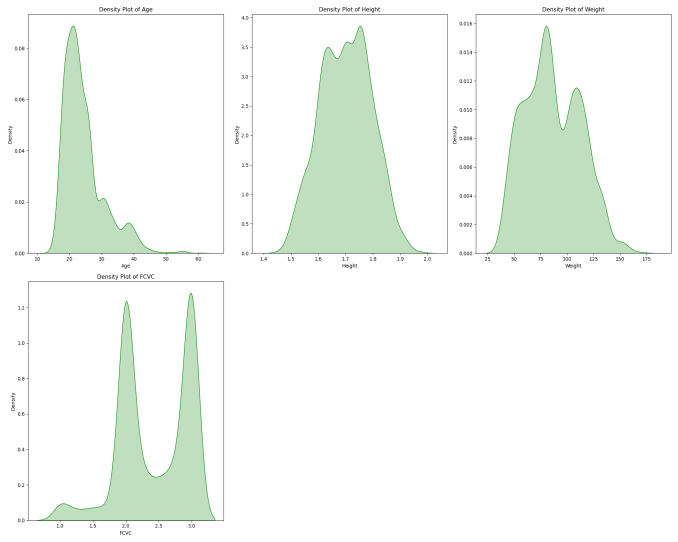


Fig. 29: Density Curve - Random Forest

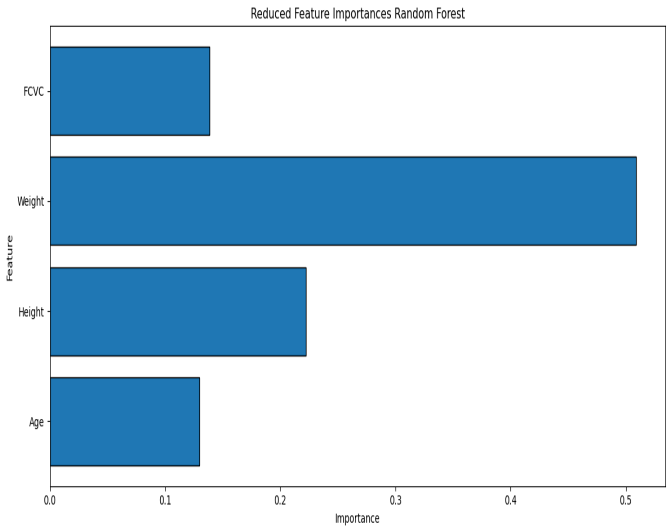


Fig. 30: Feature Importance - Random Forest

TABLE V: Cross-Validation Scores for Random Forest Model

Metric	Score
Cross-Validation Scores	[0.9078, 0.9763, 0.9787, 0.9692, 0.9787]
Mean Cross-Validation Score	0.9621

TABLE VI: Classification Report with Reduced Features (Random Forest)

Class	Precision	Recall	F1-Score	Support
0	0.96	0.96	0.96	56
1	0.95	0.89	0.92	62
2	0.97	0.97	0.97	78
3	0.98	0.97	0.97	58
4	0.98	1.00	0.99	63
5	0.90	0.98	0.94	56
6	1.00	0.98	0.99	50
Accuracy	0.96 (423 total)			
Macro Avg	0.97	0.96	0.96	423
Weighted Avg	0.97	0.96	0.96	423

F. Working Mechanism of User Interface

The User Interface (UI) for the obesity level prediction model was developed using **Streamlit**, an interactive Python library that allows users to input their data and receive predictions based on the trained machine learning model [37]. The UI integrates a pre-trained **Random Forest** classifier, which predicts obesity levels based on a set of health-related features provided by the user [39].

1) Loading the Pre-Trained Model: When the UI is launched, it automatically loads the pre-trained **Random Forest** model and a scaling method from a previously saved file. The model is essential for predicting obesity levels, while the scaler ensures that the input data is standardized in the same way the data was prepared during the training process [38].

2) Input Fields for User Data: The UI presents the user with several input fields where they can provide key health metrics:

- **Height** (in centimeters)
- **Weight** (in kilograms)
- **Age** (in years)
- **Frequency of Vegetable Consumption (FCVC)** on a scale from 1 to 3 (where, 1=never, 2=sometimes and 3=always)

These features are critical as they are used by the Random Forest model to make predictions [39]. The input fields are user-friendly, with appropriate ranges and increments, ensuring the user can enter their data with precision.

3) Validation of Inputs: Once the user has filled in the required fields, the UI validates the inputs to ensure that all necessary information has been provided [40]. If any fields are left blank or contain invalid data, the UI will prompt the user to correct the inputs before proceeding with the prediction.

4) Data Scaling: Before passing the user's input to the model, the data is scaled using the same method that was applied during the training phase [38]. This ensures that the user's input is on the same scale as the training data, enabling the model to provide accurate predictions.

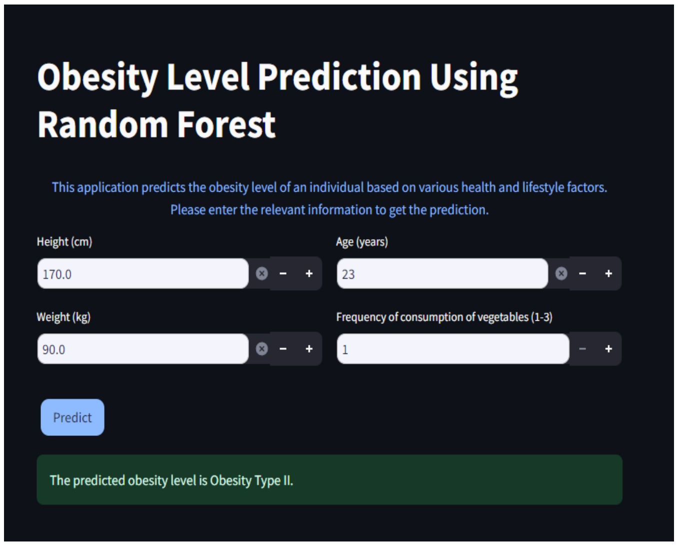
5) *Prediction Process:* After the user's input data is scaled, it is passed to the **Random Forest** model. The model analyzes the data and predicts the user's obesity level, which falls into one of several predefined categories [39], such as:

- Insufficient Weight
- Normal Weight
- Overweight Level I
- Overweight Level II
- Obesity Type I
- Obesity Type II
- Obesity Type III

6) *Displaying Results:* Once the model makes its prediction, the UI presents the result in a clear and readable format. The predicted obesity level is shown to the user as a human-readable category (e.g., "Normal Weight" or "Obesity Type I"). This makes the results easy to interpret, offering immediate feedback based on the user's inputs [41].

7) *User-Friendly Design and Interaction:* The design of the UI is made to be intuitive and visually appealing [37]. Custom styling is applied to the input fields and buttons to enhance the user experience, ensuring that the interface is not only functional but also easy to use. The interactive elements are responsive, allowing the user to enter data and receive results in real-time [37].

8) *Practical Application of Machine Learning:* The **Streamlit** interface provides a practical way for users to interact with a machine learning model without requiring any technical setup. Users can access the interface through a web browser, input their data, and immediately see the prediction results. This makes the model more accessible and demonstrates the application of machine learning in healthcare settings [37], offering users personalized insights based on their health data [39].



Obesity Level Prediction Using Random Forest

This application predicts the obesity level of an individual based on various health and lifestyle factors. Please enter the relevant information to get the prediction.

Height (cm): 170.0

Age (years): 23

Weight (kg): 90.0

Frequency of consumption of vegetables (1-3): 1

Predict

The predicted obesity level is Obesity Type II.

Fig. 31: User Interface Predicting Obisity level

G. SVM Code

```

import pandas as pd
from sklearn.model_selection import
    train_test_split, learning_curve,
    cross_val_score
from sklearn.preprocessing import LabelEncoder,
    StandardScaler
from sklearn.svm import SVC
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import accuracy_score,
    classification_report, confusion_matrix,
    roc_curve, auc, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.feature_selection import
    mutual_info_classif, VarianceThreshold,
    SelectKBest, chi2

import warnings

# Ignore all warnings
warnings.filterwarnings("ignore")


# Load the dataset
file_path = 'ObesityDataSet.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataset
print("First 5 rows of the dataset:")
print(data.head())


# Check for missing values
print("\nMissing Values in Each Column:")
print(data.isnull().sum())


# Identify categorical columns and apply Label
    Encoding
label_encoders = {}
categorical_columns =
    data.select_dtypes(include=['object']).columns

print("\nCategorical Columns:",
    list(categorical_columns))

for col in categorical_columns:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le
    print(f"Encoded '{col}' with classes:
        {le.classes_}")

print(data.head())

# Separate features (X) and the target variable
    (y)
X = data.drop('NObeyesdad', axis=1)  # Assuming
    'NObeyesdad' is the target variable
y = data['NObeyesdad']


# Calculate Mutual Information Scores
mi_scores = mutual_info_classif(X, y)
mi_scores_df = pd.DataFrame({
    'Feature': X.columns,
    'MI Score': mi_scores
}).sort_values(by='MI Score', ascending=False)

```

```python
print("\nMutual Information Scores:")
print(mi_scores_df)


# Dropping less important features based on analysis (same features as in Decision Tree and Random Forest)
features_to_drop = ['SMOKE', 'SCC', 'MTRANS', 'FAVC', 'CALC', 'TUE', 'FAF', 'CH2O', 'CAEC', 'NCP', 'FCVC', 'family_history_with_overweight','Age']
X_reduced = X.drop(columns=features_to_drop)

print("\nReduced Feature Columns:", list(X_reduced.columns))


# Feature Scaling (SVM requires scaling)
scaler = StandardScaler()
X_scaled_reduced = scaler.fit_transform(X_reduced)

# Train-Test Split with reduced feature set
X_train_reduced, X_test_reduced, y_train, y_test = train_test_split(
    X_scaled_reduced, y, test_size=0.2, random_state=42
)

print("\nTraining Set Size with Reduced Features:", X_train_reduced.shape)
print("Testing Set Size with Reduced Features:", X_test_reduced.shape)


# Train the SVM model with reduced features
clf_svm = SVC(kernel='linear', probability=True, random_state=42)
clf_svm.fit(X_train_reduced, y_train)

print("\nSVM model trained with reduced features.")


# Predictions on the test set with reduced features
y_pred_svm = clf_svm.predict(X_test_reduced)

# Accuracy
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f"\nAccuracy with reduced features (SVM): {accuracy_svm:.2f}")


# Classification Report
print("\nClassification Report with reduced features (SVM):")
print(classification_report(y_test, y_pred_svm))


# Confusion Matrix
conf_matrix_svm = confusion_matrix(y_test, y_pred_svm)

# Create the confusion matrix display
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix_svm, display_labels=label_encoders['NObeyesdad'].classes_)

# Plot with adjusted figure size and rotated x-axis labels
fig, ax = plt.subplots(figsize=(10, 7))  # Increase figure size
disp.plot(cmap=plt.cm.Blues, ax=ax, xticks_rotation=45)  # Rotate x-axis labels

plt.title("Confusion Matrix - SVM")
plt.show()

# Perform 5-fold cross-validation using the already defined model clf_svm
cv_scores = cross_val_score(clf_svm, X_scaled_reduced, y, cv=5, scoring='accuracy')

# Print the cross-validation scores
print("Cross-Validation Scores: ", cv_scores)
print("Mean Cross-Validation Score: ", cv_scores.mean())


#Train-Test Accuracy Graph
# Define the range of training sizes
train_sizes, train_scores, test_scores = learning_curve(
    clf_svm, X_train_reduced, y_train, cv=5,
    scoring='accuracy', train_sizes=np.linspace(0.1, 1.0, 10), random_state=42)

# Calculate the mean and standard deviation of the training and testing scores
train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

# Plot the learning curve
plt.figure(figsize=(10, 7))
plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.1, color="r")
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1, color="g")
plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training accuracy")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation accuracy")

plt.title("Learning Curve - SVM")
plt.xlabel("Training Size")
plt.ylabel("Accuracy")
plt.legend(loc="best")
plt.grid(True)
plt.show()


# Compute ROC curve and ROC area for each class
y_test_binarized = label_binarize(y_test, classes=np.unique(y))
n_classes = y_test_binarized.shape[1]
y_score_svm = clf_svm.predict_proba(X_test_reduced)

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ =
        roc_curve(y_test_binarized[:, i],
        y_score_svm[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve
plt.figure(figsize=(8, 6))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], lw=2,
        label=f'Class {i} (AUC =
        {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - SVM')
plt.legend(loc="lower right")
plt.show()



# Plot histograms for each numerical feature
X_reduced_df = pd.DataFrame(X_reduced,
    columns=X_reduced.columns)
X_reduced_df.hist(bins=20, figsize=(20, 15),
    color='blue', edgecolor='black')
plt.suptitle('Histograms of Features')
plt.show()


# Plot density plots for each numerical feature
plt.figure(figsize=(20, 15))
for i, col in enumerate(X_reduced_df.columns):
    plt.subplot(len(X_reduced_df.columns)//3 +
        1, 3, i + 1)
    sns.kdeplot(X_reduced_df[col], shade=True,
        color="g")
    plt.title(f'Density Plot of {col}')
plt.tight_layout()
plt.show()


# Assuming you have a trained linear SVM model
    `clf_svm`
# The coefficients are available in
    `clf_svm.coef_` for a linear kernel
coefficients = clf_svm.coef_

# If your SVM is a multi-class classifier, you
    might need to take the mean or sum of
    coefficients
# across all classes or visualize them
    separately. Here's how to handle it for
    binary classification:
importance = np.abs(coefficients).mean(axis=0)

# Create a bar plot of the feature importances
plt.figure(figsize=(12, 6))
plt.title("Feature Importance - Linear SVM")
plt.barh(X_reduced.columns, importance,
    align="center")
plt.xlabel("Coefficient Magnitude")
plt.ylabel("Feature")
plt.show()
```

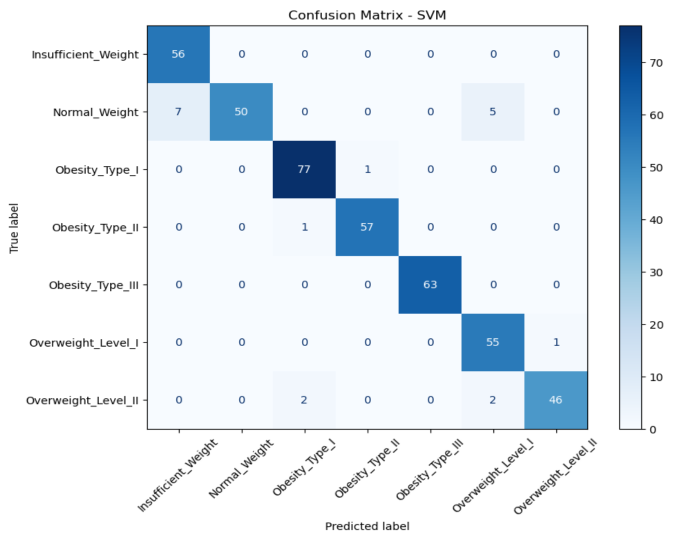


Fig. 32: Confusion Matrix: SVM

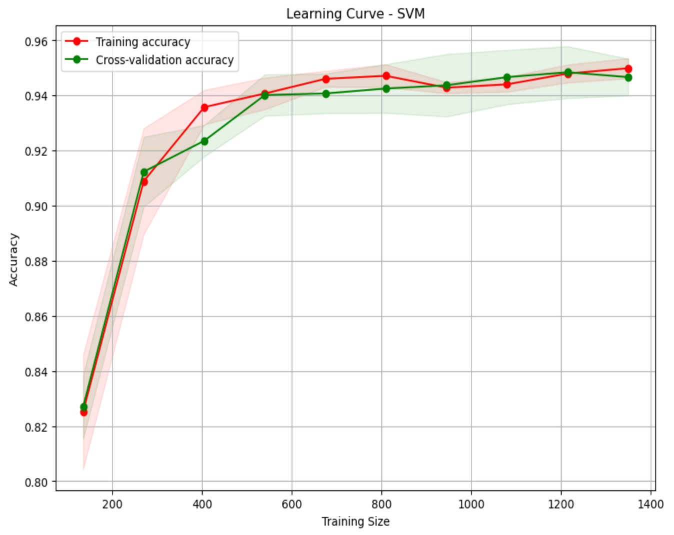


Fig. 33: Learning Curve: SVM

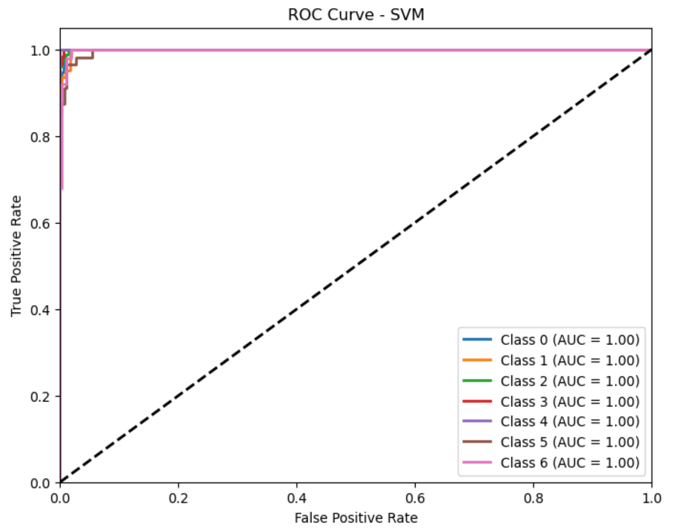


Fig. 34: ROC Curve: SVM

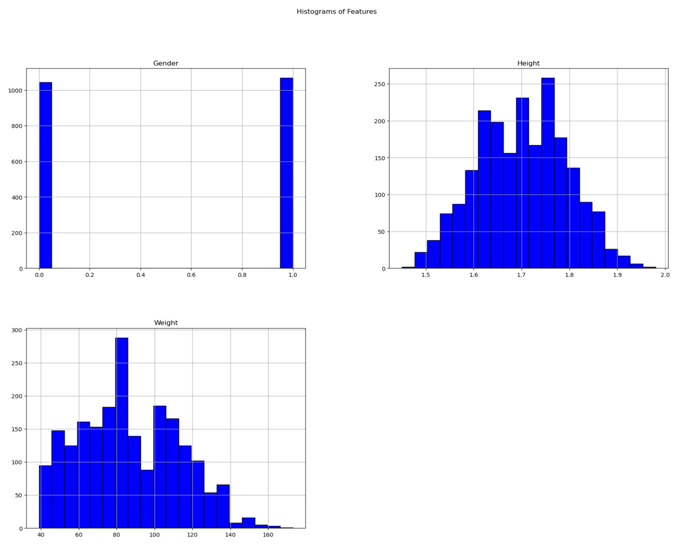


Fig. 35: Histogram: SVM

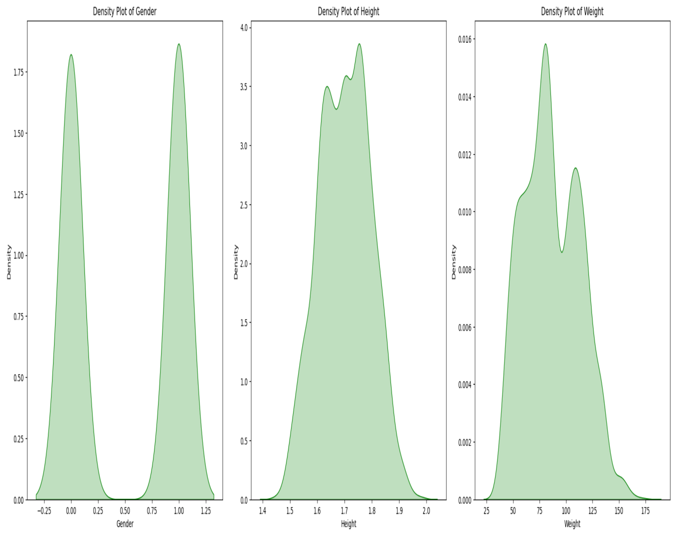


Fig. 36: Density Curve: SVM

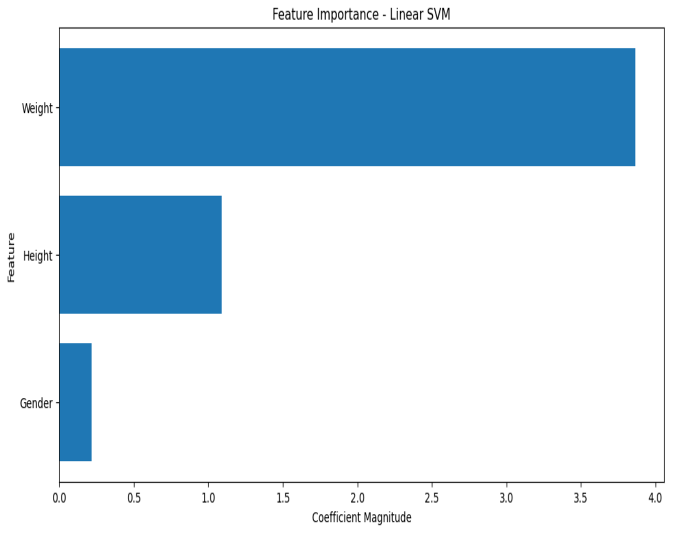


Fig. 37: Feature Importance:SVM

TABLE VII: Cross-Validation Scores and Mean Score

Metric	Score
Cross-Validation Scores	[0.9385, 0.9597, 0.9360, 0.9573, 0.9597]
Mean Cross-Validation Score	0.9503

TABLE VIII: Classification Report with Reduced Features (SVM)

Class	Precision	Recall	F1-Score	Support
0	0.89	1.00	0.94	56
1	1.00	0.81	0.89	62
2	0.96	0.99	0.97	78
3	0.98	0.98	0.98	58
4	1.00	1.00	1.00	63
5	0.89	0.98	0.93	56
6	0.98	0.92	0.95	50
Accuracy	0.96 (423 total)			
Macro Avg	0.96	0.95	0.95	423
Weighted Avg	0.96	0.96	0.95	423

PRIMARY MODEL

H. Accuracy Comparison Before Feature Selection

The following table compares the accuracy of the three models (Decision Tree, Random Forest, and Support Vector Machine) before feature selection was applied.

TABLE IX: Accuracy Comparison Before Feature Selection

Model	Accuracy (%)
Decision Tree (DT)	93%
Random Forest (RF)	96%
Support Vector Machine (SVM)	96%

As shown in the table, the Random Forest and Support Vector Machine models both achieved higher accuracy than the Decision Tree model before feature selection was performed.

I. Plots and Results before Primary Feature Selection

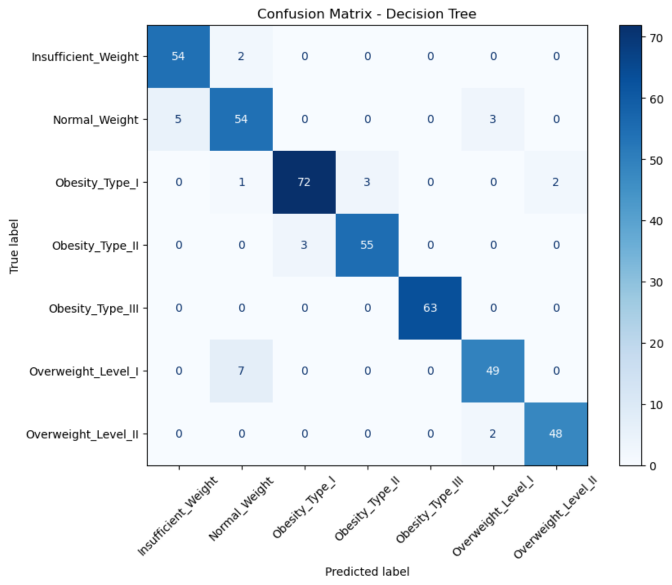


Fig. 38: Confusion Matrix: Decision Tree

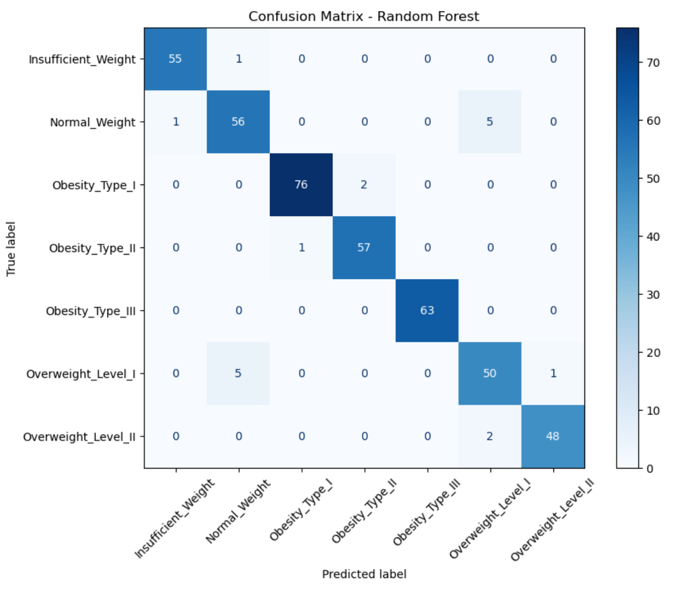


Fig. 39: Confusion Matrix: Random Forest

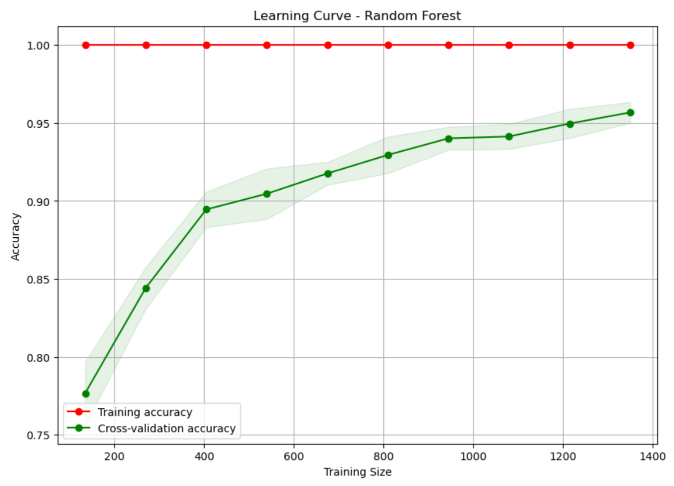


Fig. 42: Learning Curve: Random Forest

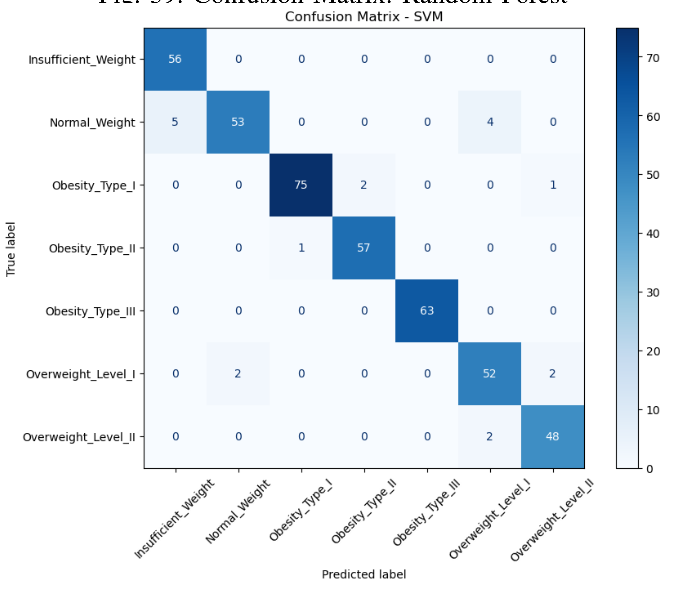


Fig. 40: Confusion Matrix: SVM

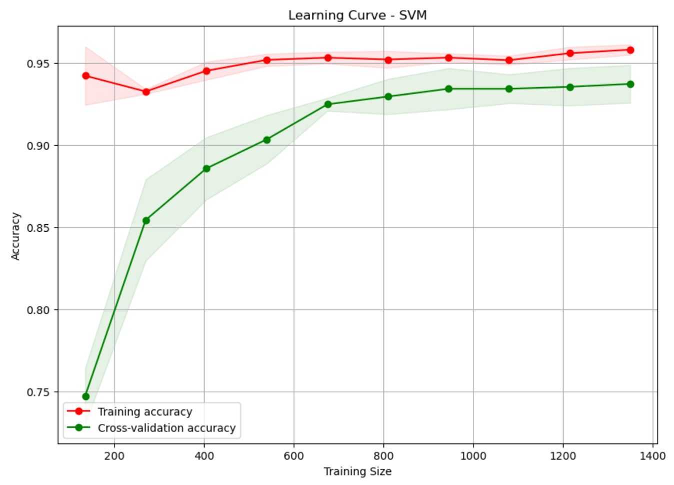


Fig. 43: Learning Curve: SVM

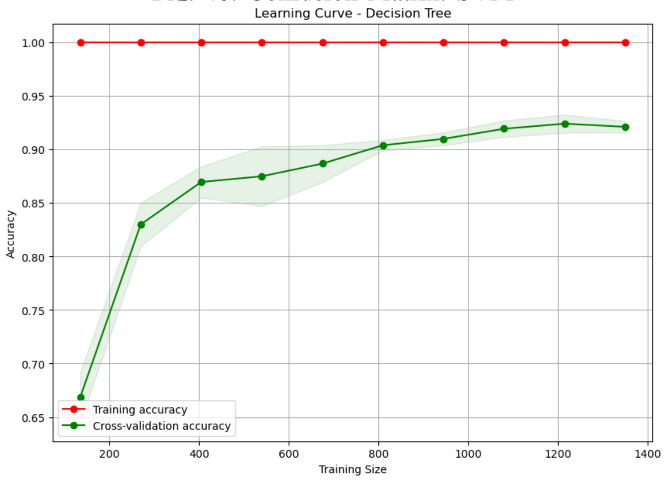


Fig. 41: Learning Curve: Decision Tree

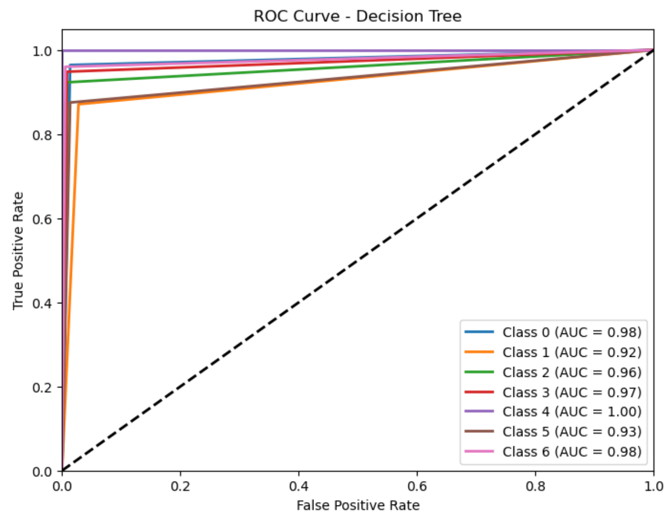


Fig. 44: ROC Curve before feature selection: Decision Tree

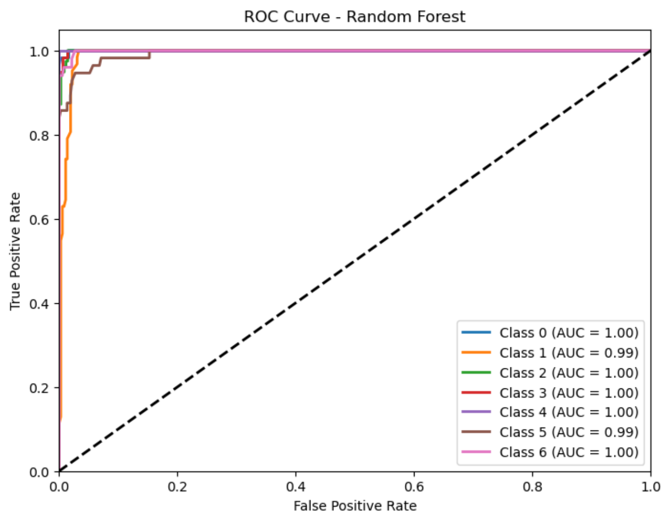


Fig. 45: ROC Curve before feature selection: Random Forest

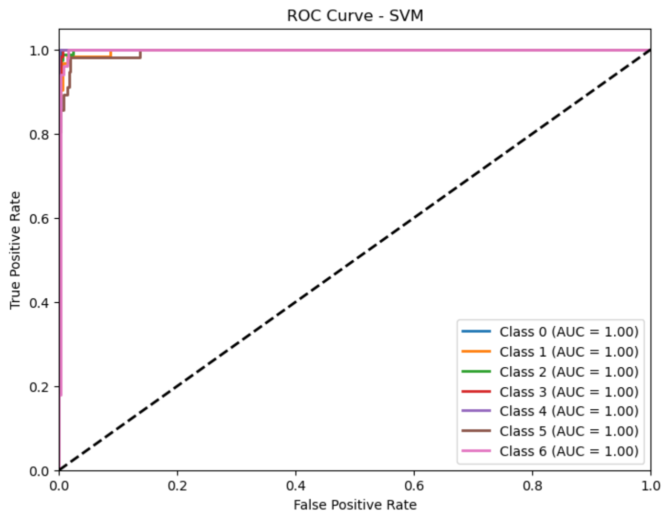


Fig. 46: ROC Curve before feature selection: SVM

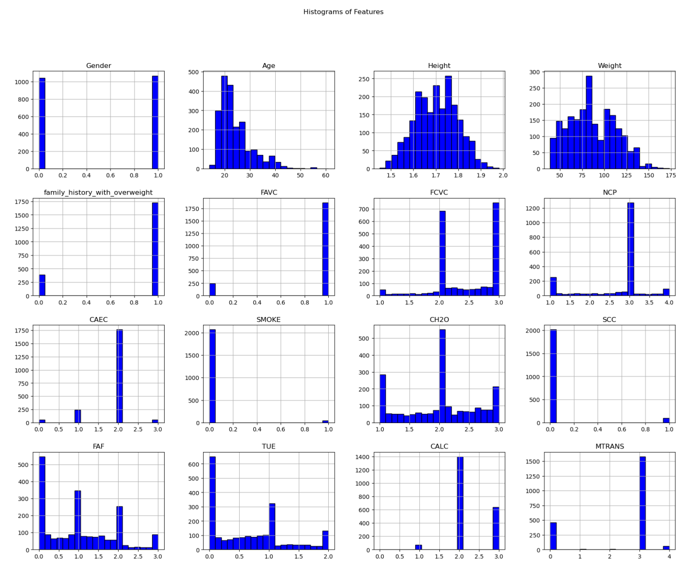


Fig. 47: Histogram of all features: All Models

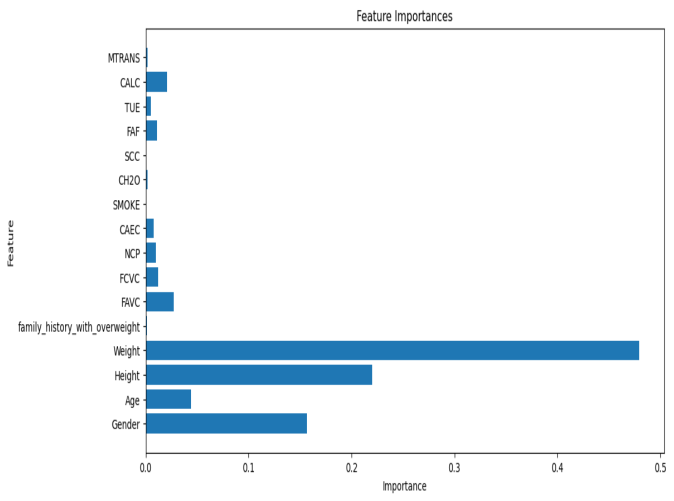


Fig. 48: Feature Importance on all features: Decision Tree

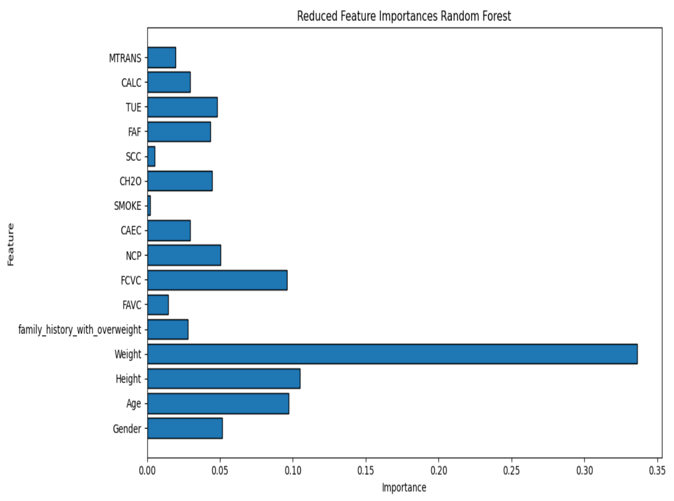


Fig. 49: Feature Importance on all features: Random Forest

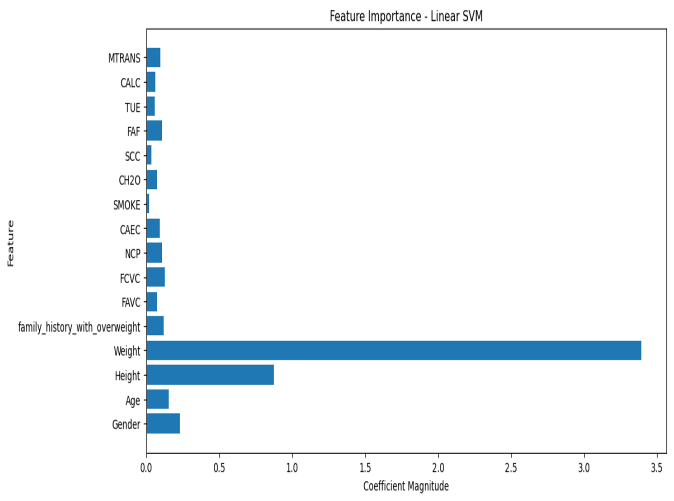


Fig. 50: Feature Importance on all features: SVM

Although the model shows high accuracy, by analyzing all the graphs and plots, it's clear that the models show misclassifications and include features with little to no contribution, such as "SCC" and "MTRANS" [42]. These irrelevant features may lead to overfitting and reduced accuracy. The models benefit significantly from feature selection, as focusing on key features like "Weight," "Height," "Age," and "Gender" enhances accuracy and generalization [43]. This highlights the importance of feature selection for improving model performance and reducing unnecessary complexity [44].

Mutual Information (MI) measures the dependency between a feature X and the target variable Y. It calculates how much information about Y is gained by knowing X [4]. A high MI score means the feature provides significant information about the target, while a low score indicates little or no useful information [35].

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right)$$

Where:

- $I(X;Y)$ is the mutual information between feature X and target Y.
- $p(x,y)$ is the joint probability of X and Y.
- $p(x)$ and $p(y)$ are the marginal probabilities of X and Y, respectively.

Mutual Information Score for the models:

TABLE X: Mutual Information Scores: Decision Tree (DT)

Feature	**MI Score**
Weight	1.252854
Age	0.576698
Height	0.418337
FCVC	0.378460
FAF	0.301072
CH2O	0.286948
TUE	0.286352
NCP	0.244932
Gender	0.181806
family_history_with_overweight	0.157534
CAEC	0.147900
CALC	0.097926
MTRANS	0.079579
FAVC	0.042003
SMOKE	0.022222
SCC	0.009348

TABLE XI: Mutual Information Scores: Random Forest (RF)

Feature	**MI Score**
Weight	1.252430
Age	0.577847
Height	0.429302
FCVC	0.385358
FAF	0.302534
TUE	0.296375
CH2O	0.294264
NCP	0.241644
Gender	0.190112
CAEC	0.175103
family_history_with_overweight	0.160716
CALC	0.086774
MTRANS	0.082284
FAVC	0.075834
SMOKE	0.042416
SCC	0.025281

TABLE XII: Mutual Information Scores: Support Vector Machine (SVM)

Feature	**MI Score**
Weight	1.253256
Age	0.579189
Height	0.422686
FCVC	0.378165
CH2O	0.289279
FAF	0.287524
NCP	0.270792
TUE	0.269207
Gender	0.198052
family_history_with_overweight	0.151779
CAEC	0.135726
CALC	0.107055
MTRANS	0.064291
FAVC	0.047166
SCC	0.041161
SMOKE	0.008124

The bottom four contributing features—SMOKE, SCC, MTRANS, and FAVC—were eliminated from all models, and the models were re-evaluated [4]. The results obtained from the three models after this adjustment are presented below.

TABLE XIII: Accuracy Comparison Before After Selection

Model	**Accuracy (%)**
Decision Tree (DT)	96%
Random Forest (RF)	95%
Support Vector Machine (SVM)	95%

After primary feature selection, the accuracy of the Random Forest and SVM models dropped to 95%, while the accuracy of the Decision Tree increased to 96%. We will also examine the learning curves of all three models to evaluate potential underfitting or overfitting issues.

J. Plots and Results after Primary Feature Selection

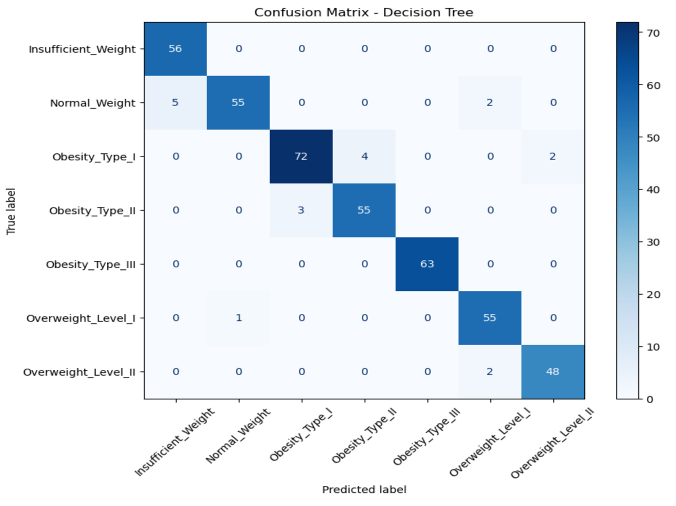


Fig. 51: Confusion Matrix After Primary Feature Selection: Decision Tree

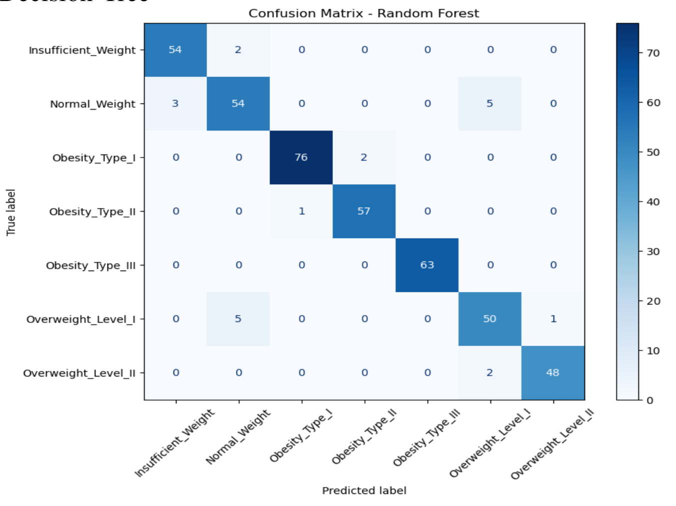


Fig. 52: Confusion Matrix After Primary Feature Selection:

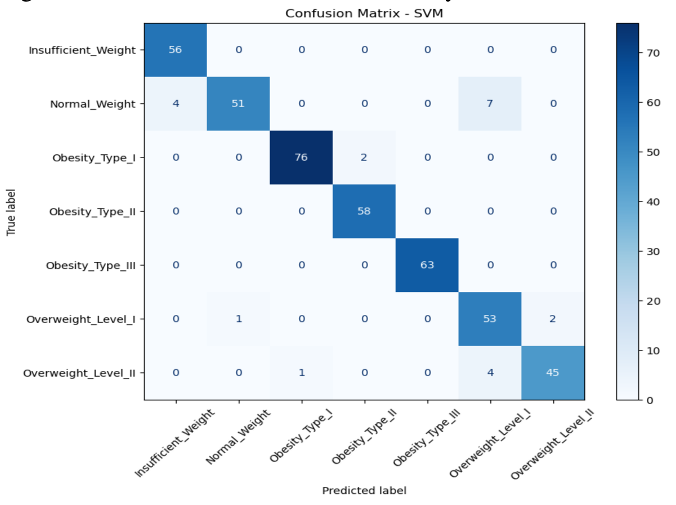


Fig. 53: Confusion Matrix After Primary Feature Selection: SVM

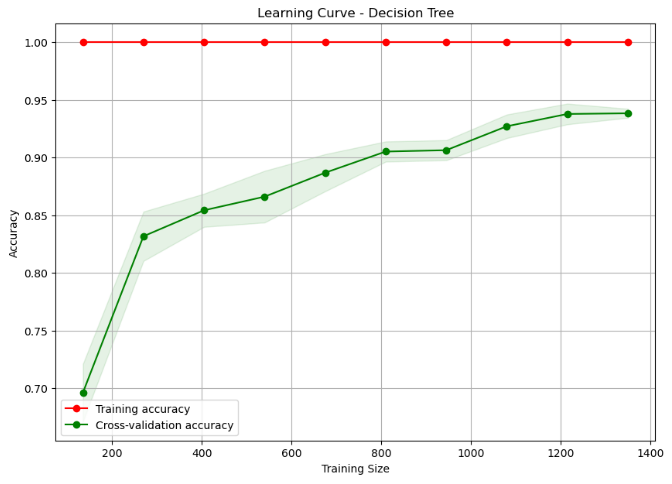


Fig. 54: Learning Curve: Decision Tree

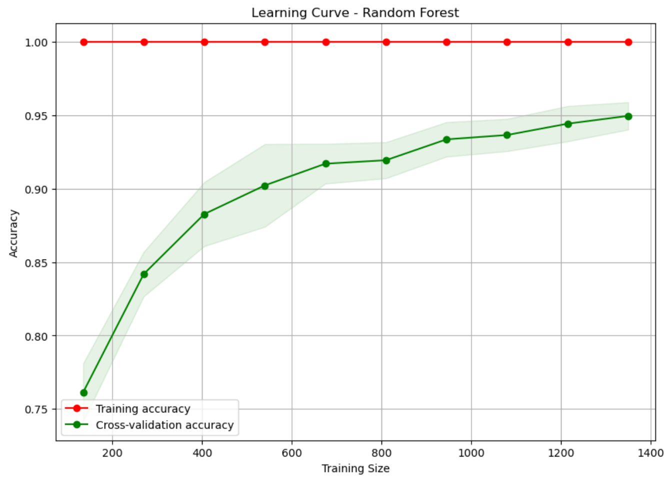


Fig. 55: Learning Curve: Random Forest

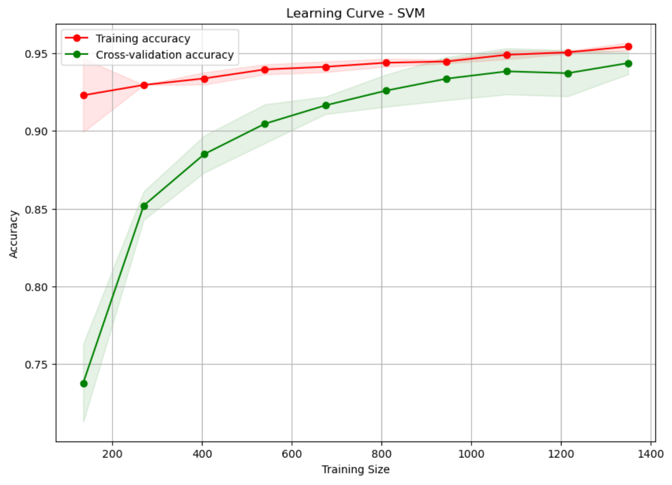


Fig. 56: Learning Curve: SVM

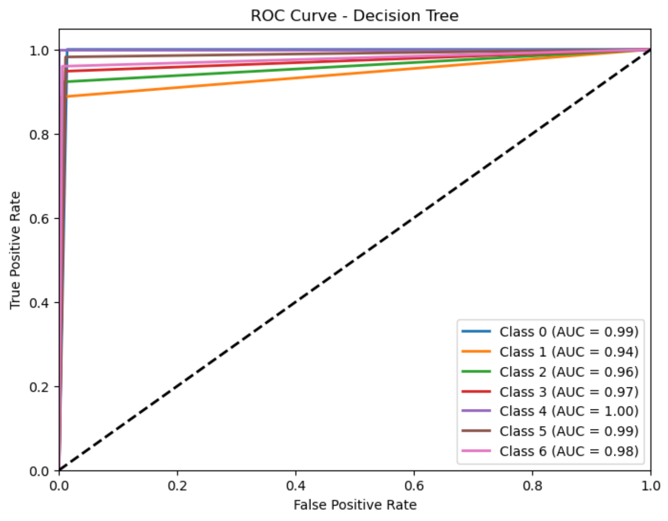


Fig. 57: ROC Curve after Primary Feature Selection: Decision Tree

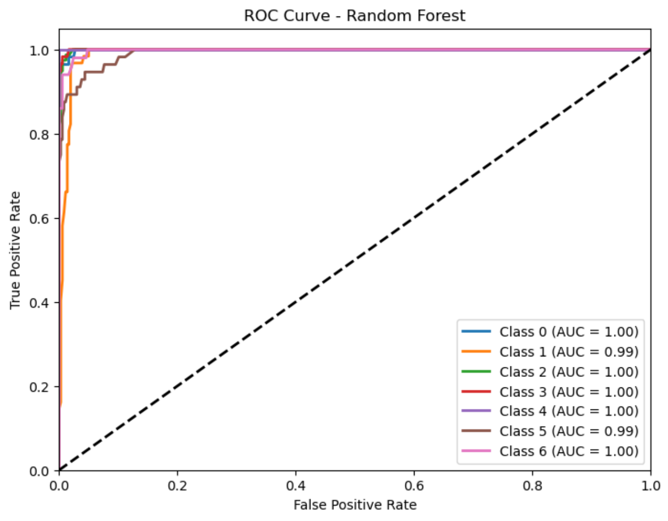


Fig. 58: ROC Curve after Primary Feature Selection: Random Forest

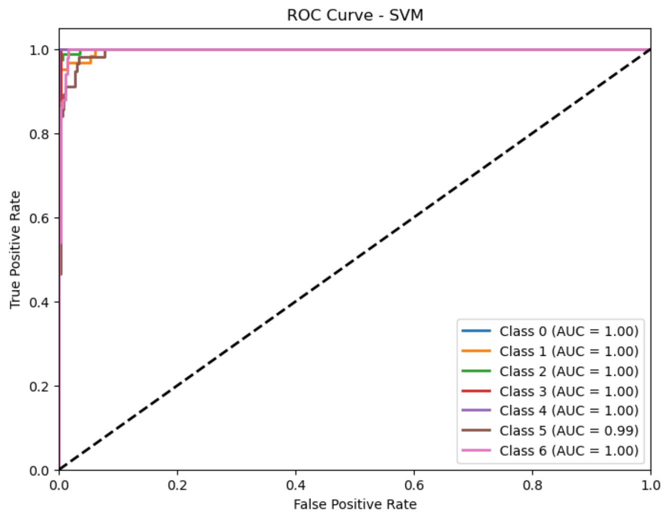


Fig. 59: ROC Curve after Primary Feature Selection: SVM

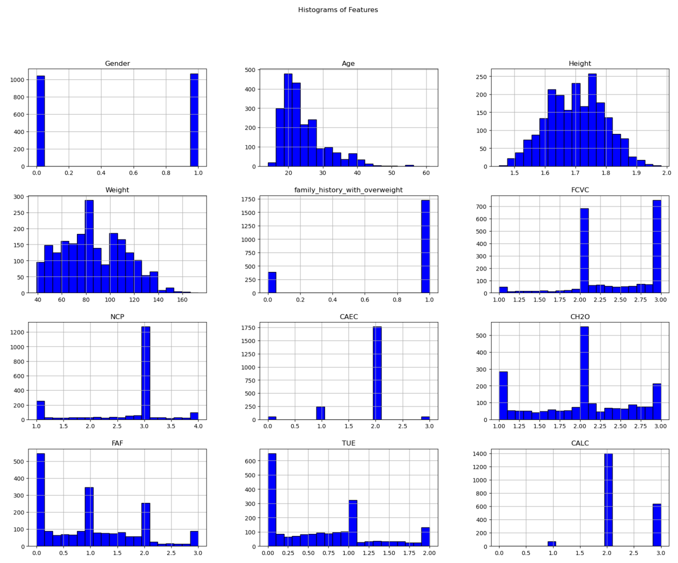


Fig. 60: Histogram after Primary Feature Selection: All Models

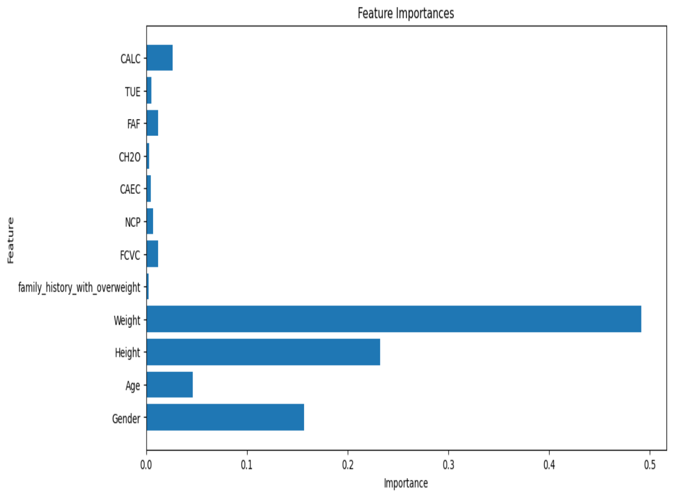


Fig. 61: Feature Importance after Primary Feature Selection: Decision Tree

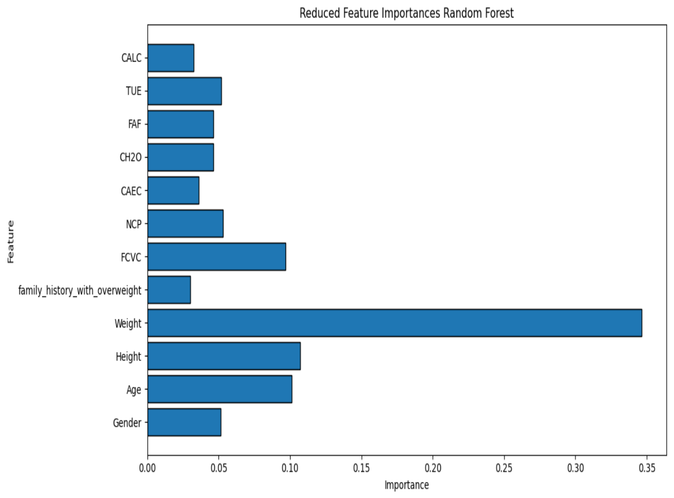


Fig. 62: Feature Importance after Primary Feature Selection: Random Forest

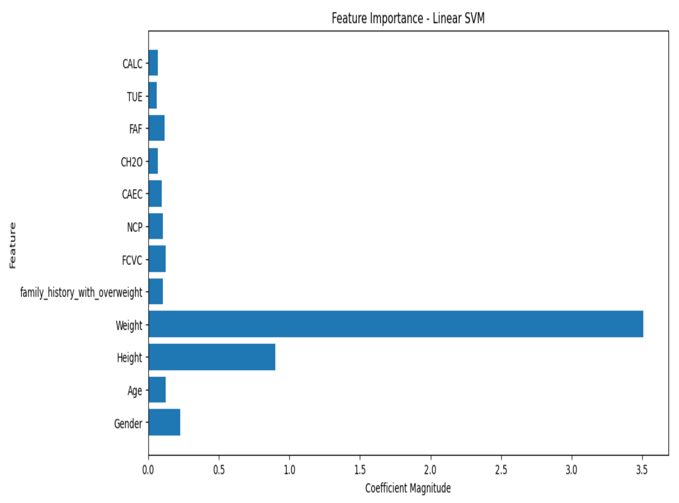


Fig. 63: Feature Importance after Primary Feature Selection: SVM

After primary feature selection, all three models—Decision Tree (DT), Random Forest (RF), and Support Vector Machine (SVM)—show different learning behaviors. The Decision Tree shows perfect training accuracy (100%) but only achieves around 90% cross-validation accuracy, indicating significant overfitting [35]. The Random Forest also has 100% training accuracy but demonstrates better generalization, with cross-validation accuracy reaching around 93-95% [39]. The SVM model, while not overfitting as much, shows a smaller gap between training (around 95%) and cross-validation accuracy (92%), indicating a more balanced model [18]. Despite these improvements, further enhancement can be made by refining hyperparameters and using regularization techniques to minimize overfitting and boost generalization performance across all models [4].

K. Accuracy Comparison After Final Feature Selection

After several iterations and improvements in feature selection using EDA and feature importance analysis, the top-performing features were chosen for each model to achieve the best accuracy. For the Decision Tree, ”Weight,” ”Height,” ”Age,” and ”Gender” were selected, while ”FCVC,” ”Weight,” ”Height,” and ”Age” were chosen for the Random Forest model. For SVM, the top three features—”Weight,” ”Height,” and ”Gender”—were used. This highlights that different models prioritize different features for training. The final comparison of the models, following this feature selection, is detailed in the main documentation provided above.

L. Comparison of SVM Kernels

Support Vector Machines (SVM) can use different kernel functions to transform input data into a higher-dimensional space, making it easier to find a separating hyperplane [18]. Below are the common types of kernels used in SVM:

- **Linear Kernel:**

$$K(x_i, x_j) = x_i^T x_j$$

The linear kernel is used when the data is linearly separable. It is the simplest and often the most efficient kernel, particularly when the dimensionality of the data does not require complex transformation [45]. Here:

  - x_i and x_j are input vectors.
  - $x_i^T x_j$ is the dot product between two vectors x_i and x_j.

- **Polynomial Kernel:**

$$K(x_i, x_j) = (x_i^T x_j + c)^d$$

The polynomial kernel is useful when the relationship between the classes is polynomial [46]. It creates a more flexible decision boundary but increases complexity. Here:

  - $x_i^T x_j$ is the dot product between two input vectors.
  - c is a constant that helps adjust the influence of higher-degree terms.
  - d is the degree of the polynomial, controlling the complexity of the decision boundary.

- **Radial Basis Function (RBF) Kernel:**

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right)$$

Also known as the Gaussian kernel, this is widely used for non-linear data. It measures the similarity between two points based on their distance, enabling more flexible decision boundaries [46]. Here:

  - $\|x_i - x_j\|$ is the Euclidean distance between the two input vectors.
  - γ is a parameter that defines how much influence a single training example has. Smaller values result in a smoother decision boundary.

- **Sigmoid Kernel:**

$$K(x_i, x_j) = \tanh(\alpha x_i^T x_j + c)$$

The sigmoid kernel is similar to the activation function of a neural network. Though effective in some cases, it is generally less favored for SVMs compared to other kernels [47]. Here:

  - α is a scaling factor.
  - $x_i^T x_j$ is the dot product between two input vectors.
  - c is a constant that controls the offset.
  - $\tanh$ is the hyperbolic tangent function, which is a bounded and continuous function.

Results obtained from using all the above-mentioned kernels are provided below.

TABLE XIV: SVM Model Performance with Different Kernels on Final Features

SVM Kernel	**Accuracy**
Linear	0.9551
Polynomial (Poly)	0.8936
RBF (Radial Basis Function)	0.9362
Sigmoid	0.6194

The table summarizes the accuracy of SVM models trained with four different kernels. The Linear kernel performed the best with an accuracy of 95.51%, followed by the RBF kernel at 93.62% [18]. The Polynomial kernel achieved 89.36%, while the Sigmoid kernel had the lowest accuracy at 61.94%, indicating it may not be ideal for this dataset [45].

M. Comparison of Accuracy on Different Train-Test Splits Across Models

TABLE XV: Train and Test Split Accuracy Across Different Models

Train-Test Split	**Decision Tree (DT)**	**Random Forest (RF)**	**SVM**
80-20 Split	96%	96%	96%
70-30 Split	94%	96%	94%
60-40 Split	96%	95%	94%

The 80-20 split was chosen as it provided the best accuracy, with all models achieving 96%, outperforming the 70-30 and 60-40 splits [49].

N. SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE is a popular technique used to handle imbalanced datasets by generating synthetic samples for the minority class [48]. It works by selecting a random sample from the minority class and creating synthetic data points between this sample and its nearest neighbors.

The formula for generating a synthetic sample is:

$$x_{\text{new}} = x_i + \lambda(x_{\text{nn}} - x_i)$$

Where:

- x_i is a minority class sample.
- x_{nn} is one of the nearest neighbors of x_i.
- λ is a random number between 0 and 1.

In this project, there was no need to apply SMOTE during the model development process because the dataset's author had already used SMOTE to balance the data [48]. This preprocessing step ensured a balanced representation of all classes, which allowed for effective model training without additional oversampling.

APPENDIX CONCLUSION

The appendix offers valuable supplementary materials that complement the main analysis, including detailed confusion matrices, feature importance plots, and extended performance metrics. These additional resources provide a more comprehensive view of the models' behaviors prior to feature selection, shedding light on the challenges encountered and the improvements made during the modeling process. By documenting the models' iterative development, the appendix highlights the critical role of exploratory data analysis (EDA) and feature selection in enhancing model accuracy and interpretability [50].

Looking forward, further improvements could be achieved through advanced techniques such as hyperparameter tuning via Grid Search or Random Search, which would help optimize model performance [49]. Applying k-fold cross-validation would provide a more robust evaluation by validating the models on multiple data splits [52]. Incorporating ensemble methods, such as stacking or blending, may further improve accuracy by leveraging the strengths of multiple models [51]. Additionally, regularization techniques (e.g., L1/L2) can help mitigate overfitting, and more sophisticated feature engineering may reveal complex patterns in the data [50]. Addressing class imbalances with methods like SMOTE and testing on additional datasets would enhance model generalization. Finally, exploring deep learning approaches could yield improved results, especially for more intricate or large-scale problems [53].