STAT 627/427 Statistical Machine Learning - Final Project

Meng Hsuan Ho, Pin Tzu Tseng, Dongni Li

2025-04-27

Introduction

Data Source: [Obesity Levels] from UCI Machine Learning Repository (https://archive.ics.uci.edu/dataset/544/estimation+of+obesity+levels+based+on+eating+habits+and+physical+condition)

Statement and Importance: This study builds predictive models to classify obesity levels. It contributes to public health by offering a tool for early identification of individuals at risk of obesity-related health issues. In real-world applications, such models can support preventive healthcare strategies. They can also personalize lifestyle interventions and optimize the allocation of healthcare resources.

Data Size: 2111 samples, 17 columns

Numerical Features

- **Age:** Right-skewed distribution, concentrated around 20 years old. Mostly young individuals with a small proportion of middle-aged individuals.
- **Weight:** Relatively dispersed with a slight right skew. Different clusters of weight groups are visible.
- **Height:** Approximately normal distribution, mainly concentrated between 1.65m and 1.85m.
- **CH2O (Water intake):** Distinct peaks at 2 and 3 liters. Significant differences in daily water intake habits among individuals.
- **FAF (Physical activity frequency):** Discrete distribution with multiple peaks. Many individuals either have very low activity or specific exercise habits.
- **TUE (Technology use time):** Discrete distribution. Most individuals have low usage, with a group of moderate users.
- **FCVC (Vegetable consumption frequency):** Clear peaks at 2 and 3. Many individuals frequently consume vegetables but the distribution deviates from normality.
- **NCP (Number of meals per day):** Highly concentrated at three meals per day. Very few extreme values.

Categorical Features

- **Gender:** Balanced distribution between male and female.
- **Family History with Overweight:** Majority have a family history of overweight.
- **FAVC (Frequent consumption of high-calorie food):** Majority answered “Yes”, indicating frequent consumption.
- **SMOKE:** Vast majority answered “No”, meaning very few smokers.
- **SCC (Monitoring calorie intake):** Vast majority answered “No”, meaning most people do not monitor their intake.
- **CALC (Alcohol consumption frequency):** Most people chose “Sometimes”, with three distinct categories observed.
- **CAEC (Consumption between meals):** Majority answered “Sometimes”, indicating an uneven distribution.
- **MTRANS (Transportation mode):** Highly concentrated in “Public Transportation”.

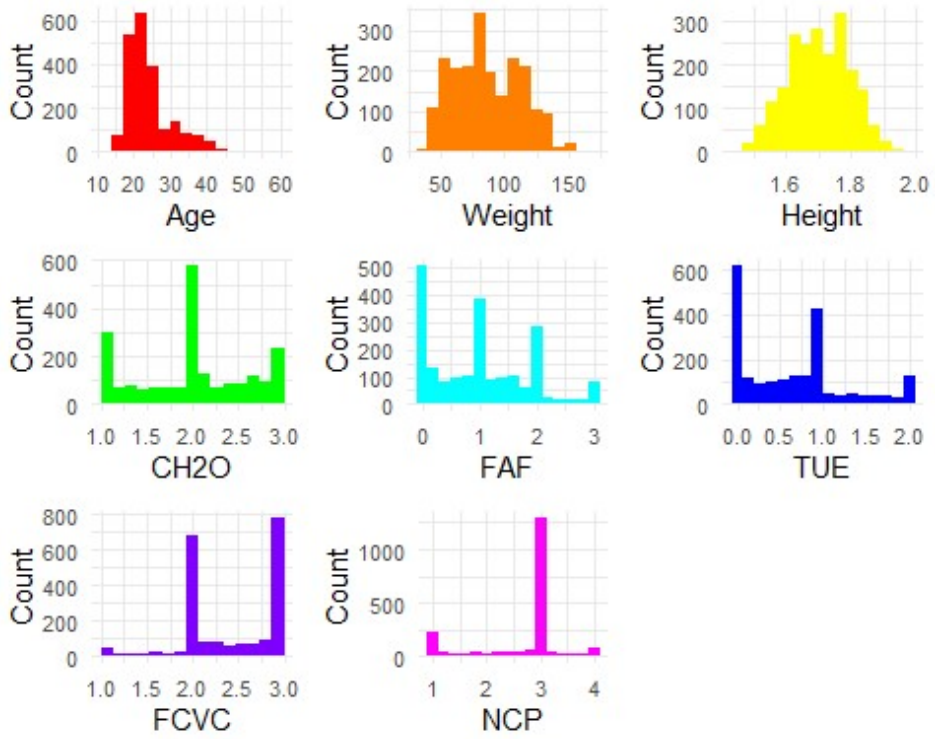
EDA (Contribute: Dongni Li, Pin Tzu Tseng)

Prior to training classification models, we conducted exploratory data analysis (EDA) to examine the distribution and structure of all features. The dataset includes a mix of numerical and categorical variables. Each exhibiting unique distributional characteristics that influence preprocessing decisions.

```
cat("Missing Value:", sum(is.na(df)), "\n")
```

```
Missing Value: 0
```

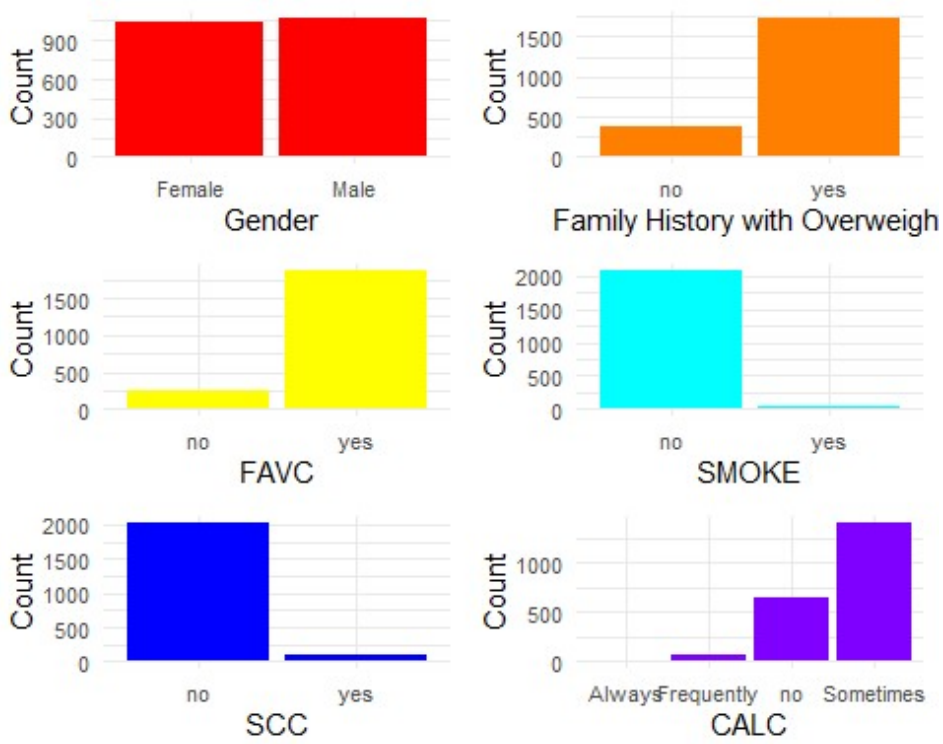
```
# Set ggplot2 plotting theme
theme_set(theme_minimal())

# Create a histogram for each numeric feature
p1 <- ggplot(df, aes(x = Age)) + geom_histogram(bins = 16, fill = "#ff0000") + theme(axis.
text.x = element_text(size = 8), axis.text.y = element_text(size = 8)) + xlab("Age") + yl
ab("Count")
p2 <- ggplot(df, aes(x = Weight)) + geom_histogram(bins = 16, fill = "#ff8000") + theme(a
xis.text.x = element_text(size = 8), axis.text.y = element_text(size = 8)) + xlab("Weight
") + ylab("Count")
p3 <- ggplot(df, aes(x = Height)) + geom_histogram(bins = 16, fill = "#ffff00") + theme(a
xis.text.x = element_text(size = 8), axis.text.y = element_text(size = 8)) + xlab("Height
") + ylab("Count")
p4 <- ggplot(df, aes(x = CH2O)) + geom_histogram(bins = 16, fill = "#00ff00") + theme(axi
s.text.x = element_text(size = 8), axis.text.y = element_text(size = 8)) + xlab("CH2O") +
 ylab("Count")
p5 <- ggplot(df, aes(x = FAF)) + geom_histogram(bins = 16, fill = "#00ffff") + theme(axis.
text.x = element_text(size = 8), axis.text.y = element_text(size = 8)) + xlab("FAF") + yl
ab("Count")
p6 <- ggplot(df, aes(x = TUE)) + geom_histogram(bins = 16, fill = "#0000ff") + theme(axis.
text.x = element_text(size = 8), axis.text.y = element_text(size = 8)) + xlab("TUE") + yl
ab("Count")
p7 <- ggplot(df, aes(x = FCVC)) + geom_histogram(bins = 16, fill = "#8000ff") + theme(axi
s.text.x = element_text(size = 8), axis.text.y = element_text(size = 8)) + xlab("FCVC") +
 ylab("Count")
p8 <- ggplot(df, aes(x = NCP)) + geom_histogram(bins = 16, fill = "#ff00ff") + theme(axis.
text.x = element_text(size = 8), axis.text.y = element_text(size = 8)) + xlab("NCP") + yl
ab("Count")
p9 <- ggplot() + theme_void()  # Blank area, remove axis labels

# Arrange plots into a 3x3 grid layout
grid.arrange(p1, p2, p3, p4, p5, p6, p7, p8, p9, ncol = 3, nrow = 3)
```



Numerical Features:

Eight continuous variables: Age, Weight, Height, CH2O, FAF, TUE, FCVC, and NCP display varying degrees of skewness and multimodality. For instance, Age and Weight are moderately right-skewed while Height is approximately normally distributed. Variables like CH2O, FAF, and NCP exhibit spikes due to discretized or self-reported input values. To ensure compatibility with models sensitive to feature scaling such as Logistic Regression with Lasso, LDA, KNN, and SVM, **all numerical features were standardized using z-score normalization** (mean = 0, standard deviation = 1). This step is essential to prevent features with larger ranges from dominating the model's optimization process. Notably, Decision Trees do not require standardization due to their non-parametric, split-based nature.

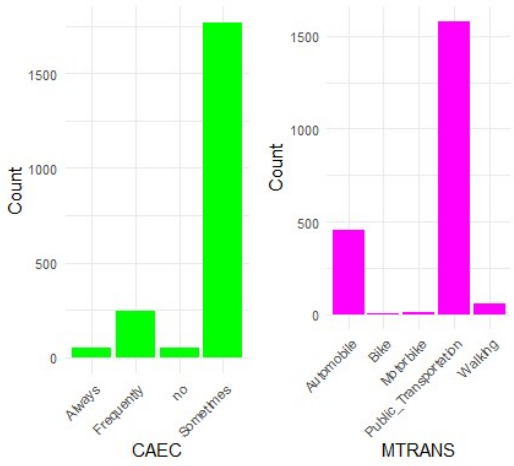
```
# Set ggplot2 theme again (if reused later)
theme_set(theme_minimal())

# Create a bar chart for a categorical feature
p1 <- ggplot(df, aes(x = Gender)) +geom_bar(fill = "#ff0000") +theme(axis.text.x = elemen
t_text(size = 8), axis.text.y = element_text(size = 8)) +xlab("Gender") + ylab("Count")
p2 <- ggplot(df, aes(x = family_history_with_overweight)) +geom_bar(fill = "#ff8000") +th
eme(axis.text.x = element_text(size = 8), axis.text.y = element_text(size = 8)) +xlab("Fa
mily History with Overweight") + ylab("Count")
p3 <- ggplot(df, aes(x = FAVC)) +geom_bar(fill = "#ffff00") +theme(axis.text.x = element_
text(size = 8), axis.text.y = element_text(size = 8)) +xlab("FAVC") + ylab("Count")
p4 <- ggplot(df, aes(x = SMOKE)) +geom_bar(fill = "#00ffff") +theme(axis.text.x = element
_text(size = 8), axis.text.y = element_text(size = 8)) +xlab("SMOKE") + ylab("Count")
p5 <- ggplot(df, aes(x = SCC)) +geom_bar(fill = "#0000ff") +theme(axis.text.x = element_t
ext(size = 8), axis.text.y = element_text(size = 8)) +xlab("SCC") + ylab("Count")
p6 <- ggplot(df, aes(x = CALC)) +geom_bar(fill = "#8000ff") +theme(axis.text.x = element_
text(size = 8), axis.text.y = element_text(size = 8)) +xlab("CALC") + ylab("Count")


# Arrange plots into a 3-row, 2-column grid layout
grid.arrange(p1, p2, p3, p4, p5, p6, ncol = 2, nrow = 3)
```



Categorical Features:

The dataset contains multiple categorical variables including Gender, Family History with Overweight, FAVC, SMOKE, SCC, CALC, CAEC, and MTRANS. Several of these variables such as CALC, CAEC, and MTRANS, contain more than two levels and exhibit strong class imbalances. To facilitate their use in machine learning models, these features **were converted into binary dummy variables via one-hot encoding.** This transformation is required for Logistic Regression, LDA, KNN, and SVM, which operate on numeric inputs. On the other hand, Decision Trees can natively handle categorical features if they are encoded as factors.

```
theme_set(theme_minimal())

p1 <- ggplot(df, aes(x = CAEC)) +geom_bar(fill = "#00ff00") +theme(axis.text.x = element_
text(angle = 45, hjust = 1, size = 8), axis.text.y = element_text(size = 8)) +xlab("CAEC")
 + ylab("Count")
p2 <- ggplot(df, aes(x = MTRANS)) +geom_bar(fill = "#ff00ff") +theme(axis.text.x = elemen
t_text(angle = 45, hjust = 1, size = 8), axis.text.y = element_text(size = 8)) +xlab("MTR
ANS") + ylab("Count")
p3 <- ggplot() + theme_void()

# Set width and height proportions for the plot layout
# Set widths and heights to match the number of rows and columns
grid.arrange(p1, p2, p3,
             ncol = 3,
             nrow = 1,
             widths = c(4, 4, 1))   # Control the width of each column
```



Constant or Near-Constant Features:

During preprocessing, we identified one-hot encoded variables such as **CALC.Always and SMOKE.yes** that exhibited **no or very little variation**. These constant features provide no information gain and may cause errors during scaling.

Data Preprocessing (Contribute: Meng Hsuan Ho)

```
# Step 1: Categorical Variable Encoding
# Identify categorical variables
categorical_vars <- c("Gender", "family_history_with_overweight", "FAVC","CAEC", "SMOKE",
 "SCC", "CALC", "MTRANS")
# Convert these columns to factors
df[categorical_vars] <- lapply(df[categorical_vars], as.factor)
# One-hot encode categorical variables (excluding the target variable 'NObeyesdad')
dummies <- dummyVars(~ ., data = df[, categorical_vars])
categorical_data <- predict(dummies, newdata = df[, categorical_vars])
categorical_data <- as.data.frame(categorical_data)

# Step 2: Numerical Variable Standardization
# Identify numerical variables
numerical_vars <- c("Age", "Height", "Weight", "FCVC", "NCP", "CH2O", "FAF", "TUE")
# Standardize numerical features
numerical_data <- scale(df[, numerical_vars])
numerical_data <- as.data.frame(numerical_data)

# Step 3: Combine Preprocessed Features + Target Variable
# Combine standardized numerical data and one-hot encoded categorical data
processed_data <- cbind(numerical_data, categorical_data)
# Add the target variable 'NObeyesdad' back
processed_data$NObeyesdad <- as.factor(df$NObeyesdad)

# Step 4: Train/Test Split
set.seed(123)
# Stratified sampling based on the target variable 'NObeyesdad'
train_index <- createDataPartition(processed_data$NObeyesdad, p = 0.8, list = FALSE)
# Create training and testing sets
train_data <- processed_data[train_index, ]
test_data  <- processed_data[-train_index, ]
```

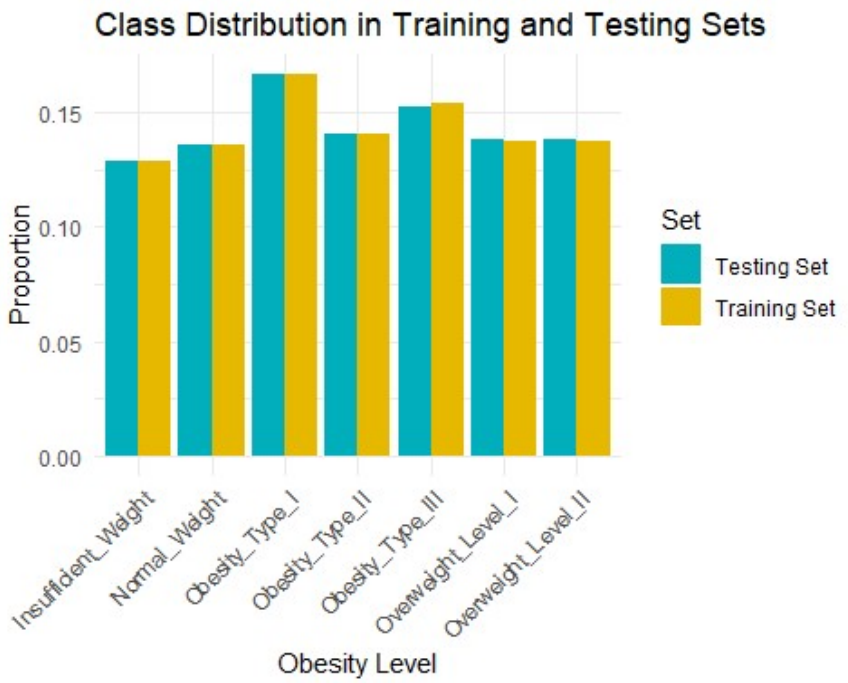
```r
# Confirm the split result
cat("Training set size:", nrow(train_data), "\n")
```

```
Training set size: 1691
```

```r
cat("Testing set size:", nrow(test_data), "\n")
```

```
Testing set size: 420
```

```r
# Check the class distribution in training and testing sets
# Calculate class distribution for the training set
train_dist <- as.data.frame(prop.table(table(train_data$NObeyesdad)))
colnames(train_dist) <- c("Class", "Proportion")
train_dist$Set <- "Training Set"

# Calculate class distribution for the testing set
test_dist <- as.data.frame(prop.table(table(test_data$NObeyesdad)))
colnames(test_dist) <- c("Class", "Proportion")
test_dist$Set <- "Testing Set"

# Combine the two datasets
combined_dist <- rbind(train_dist, test_dist)

# Plot the class distributions
ggplot(combined_dist, aes(x = Class, y = Proportion, fill = Set)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Class Distribution in Training and Testing Sets",
       x = "Obesity Level",
       y = "Proportion") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_fill_manual(values = c("#00AFBB", "#E7B800"))
```



- **Final Dataset: processed_data**
- **Final train/test data: train_data, test_data**

Modeling Approach

To evaluate the predictive performance of various classification algorithms on obesity level prediction, we implemented five supervised learning models: **Linear Discriminant Analysis (LDA), K-Nearest Neighbors (KNN), Decision Trees, Support Vector Machines (SVM), and Logistic Regression with Lasso regularization**. Each model was selected to represent a different family of classifiers with varying assumptions and inductive biases.

We used the **preprocessed dataset (processed_data)**, which includes standardized numerical features and one-hot encoded categorical variables. The dataset was partitioned into a **training set (80%)** and a **testing set (20%)** using stratified sampling to preserve the proportional distribution of obesity classes.

Linear Discriminant Analysis (LDA) (Contribute: Pin Tzu Tseng)

```
#use original dataset to split the data into train and test
set.seed(123)
# Stratified sampling based on the target variable 'NObeyesdad'
df_train_index <- createDataPartition(df$NObeyesdad, p = 0.8, list = FALSE)
# Create training and testing sets
df_train_data <- df[train_index, ]
df_test_data  <- df[-train_index, ]
# Confirm the split result
cat("df Training set size:", nrow(df_train_data), "\n")
```

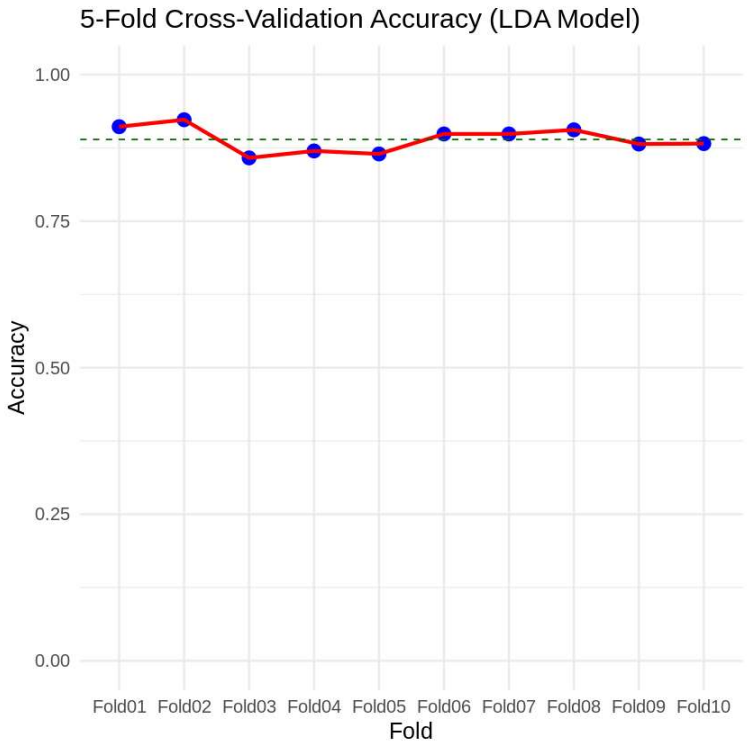
```
df Training set size: 1691
```

```
cat("df Testing set size:", nrow(df_test_data), "\n")
```

```
df Testing set size: 420
```

```
#LDA
set.seed(123)

# Create 5-fold cross-validation control settings
cv_control <- trainControl(method = "cv", number = 5)

# Train LDA model using 5-fold cross-validation via caret::train()
lda_cv_model <- train(
  NObeyesdad ~ .,
  data = df_train_data,
  method = "lda",
  trControl = cv_control)

#Extract resampling results
cv_results <- lda_cv_model$resample

#Plot the cross-validation accuracy for each fold
ggplot(cv_results, aes(x = as.factor(Resample), y = Accuracy)) +
  geom_point(size = 4, color = "blue") +
  geom_line(aes(group = 1), linewidth = 1.2, color = "red") +
  geom_hline(
    yintercept = mean(cv_results$Accuracy),linetype = "dashed", color = "darkgreen")+
  labs(title = "5-Fold Cross-Validation Accuracy (LDA Model)",
       x = "Fold",
```

```
       y = "Accuracy") +
  ylim(0,1) +
  theme_minimal(base_size = 15)
```

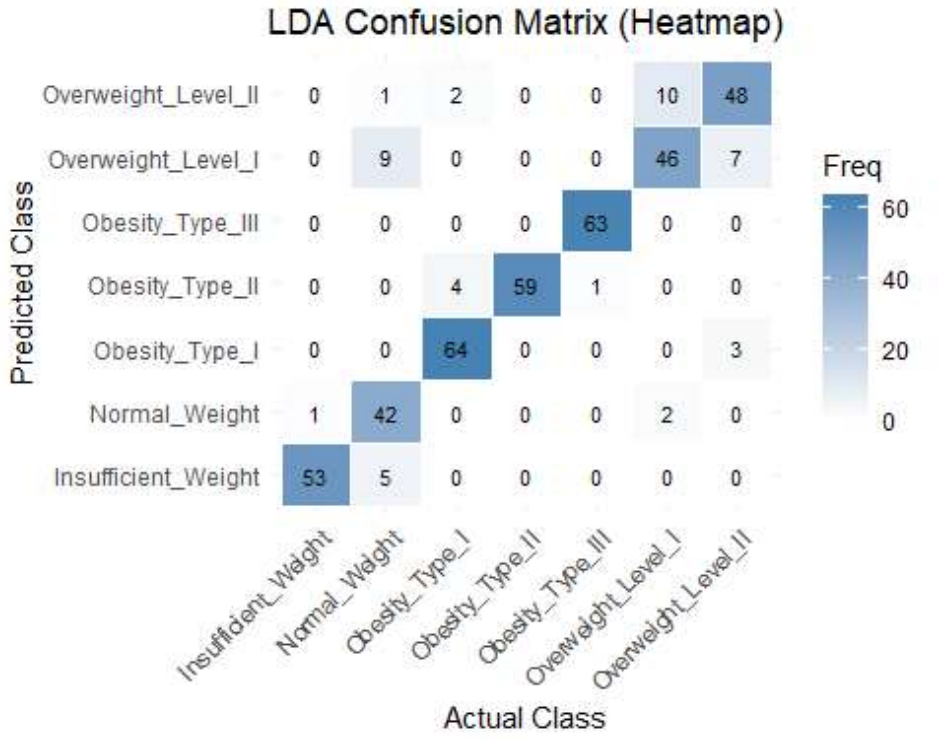



```
# Make final predictions on the test set
lda_test_pred <- predict(lda_cv_model, newdata = df_test_data)

# Calculate accuracy on the test set
accuracy <- mean(lda_test_pred == df_test_data$NObeyesdad)
cat("LDA Test Set Accuracy (after 5-fold CV training):", round(accuracy, 4), "\n")
```

```
LDA Test Set Accuracy (after 5-fold CV training): 0.8929
```

```
# Make sure both are factors
predicted_factor <- factor(lda_test_pred)
actual_factor <- factor(df_test_data$NObeyesdad)

# Then compute the confusion matrix
lda_conf_matrix <- confusionMatrix(predicted_factor, actual_factor)

# Convert the table to a data frame (long format for heatmap)
cm_df <- as.data.frame(lda_conf_matrix$table)
colnames(cm_df) <- c("Predicted", "Actual", "Freq")

# Plot confusion matrix heatmap
ggplot(cm_df, aes(x = Actual, y = Predicted, fill = Freq)) +
  geom_tile(color = "white") +
  geom_text(aes(label = Freq), color = "black", size = 3) +
  scale_fill_gradient(low = "white", high = "steelblue") +
  labs(
    title = "LDA Confusion Matrix (Heatmap)",
    x = "Actual Class",
```

```
    y = "Predicted Class"
  ) +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1),
    plot.title = element_text(hjust = 0.5))
```



The Linear Discriminant Analysis (LDA) model was trained using **5-fold cross-validation** to assess its generalization performance. As shown in the cross-validation accuracy plot, the model achieved consistently high performance across all folds with accuracies ranging from approximately 0.86 to 0.92. This relatively narrow fluctuation suggests that **the model is stable** and not overly sensitive to specific train-test splits. The average performance across folds supports the reliability of LDA in this multiclass classification task.

After cross-validation, the final LDA model was evaluated on the test set, achieving an **accuracy of 0.8929**, indicating strong predictive capability. The **confusion matrix** further highlights the model's strengths and weaknesses. The model performed exceptionally well on well-separated classes such as Obesity_Type_I, Obesity_Type_II, and Obesity_Type_III, each with high true positive counts and minimal misclassifications.

However, the model exhibited challenges distinguishing between neighboring weight categories particularly Insufficient_Weight and Normal_Weight, as well as Overweight_Level_I and Overweight_Level_II. These misclassifications likely stem from overlapping feature distributions in those classes which is a known limitation of linear classifiers when dealing with subtle class boundaries.

Overall, the LDA model demonstrated both high accuracy and consistent performance.

K-Nearest Neighbors (KNN) (Contribute: Pin Tzu Tseng)

```
# Split into features (X) and labels (y)
set.seed(123)
X_train <- subset(train_data, select = -NObeyesdad)
y_train <- train_data$NObeyesdad

X_test  <- subset(test_data, select = -NObeyesdad)
y_test  <- test_data$NObeyesdad

cat("Training set size (X):", nrow(X_train), "rows,", ncol(X_train), "columns\n")
```

```
Training set size (X): 1691 rows, 31 columns
```

```
cat("Testing set size (X):", nrow(X_test), "rows,", ncol(X_test), "columns\n")
```

```
Testing set size (X): 420 rows, 31 columns
```

```
cat("Training set size (y):", length(y_train), "\n")
```

```
Training set size (y): 1691
```

```
cat("Testing set size (y):", length(y_test), "\n")
```

```
Testing set size (y): 420
```

```
# 5-fold Cross-Validation
# KNN: Train and Test Accuracy by k
set.seed(123)

# Create 5-fold cross-validation folds
folds <- createFolds(y_train, k = 5, list = TRUE, returnTrain = FALSE)

# Initialize vectors to store accuracies
cv.accuracy = rep(0, 50)     # Cross-validation accuracy
train.accuracy = rep(0, 50)  # Training set accuracy

# Loop over k = 1 to 50
for (k in 1:50) {
  fold_accuracies = c()

  for (i in 1:5) {
    # Define training and validation sets
    validation_indices <- folds[[i]]
    X_cv_train <- X_train[-validation_indices, ]
    y_cv_train <- y_train[-validation_indices]
    X_cv_valid <- X_train[validation_indices, ]
    y_cv_valid <- y_train[validation_indices]

    # KNN prediction
    KNN_cv = knn(train = X_cv_train, test = X_cv_valid, cl = y_cv_train, k = k)

    # Calculate accuracy for this fold
    fold_accuracy = mean(KNN_cv == y_cv_valid)
    fold_accuracies = c(fold_accuracies, fold_accuracy)
  }
```

```r
  # Average accuracy over 5 folds
  cv.accuracy[k] = mean(fold_accuracies)

  # Train on entire training set and predict itself to get train accuracy
  KNN_train = knn(train = X_train, test = X_train, cl = y_train, k = k)
  train.accuracy[k] = mean(KNN_train == y_train)
}

# Find the best k based on cross-validation
best_k <- which.max(cv.accuracy)
best_cv_accuracy <- max(cv.accuracy)

cat("Best k (5-fold CV):", best_k, "\n")
```

```
Best k (5-fold CV): 1
```

```r
cat("Best CV Accuracy:", round(best_cv_accuracy, 4), "\n")
```

```
Best CV Accuracy: 0.848
```

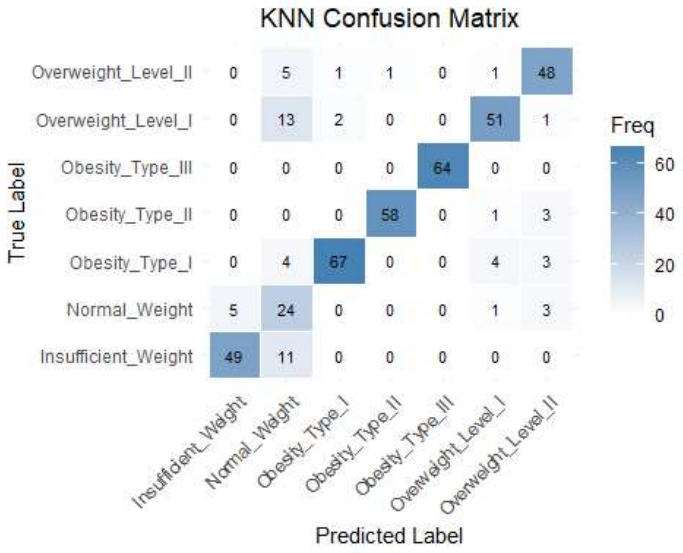
```r
# Train final model with best k on entire training set and test on test set
final_knn_pred <- knn(train = X_train, test = X_test, cl = y_train, k = best_k)

# Evaluate final model on test set
final_test_accuracy <- mean(final_knn_pred == y_test)
cat("Test Set Accuracy with Best k:", round(final_test_accuracy, 4), "\n")
```

```
Test Set Accuracy with Best k: 0.8595
```

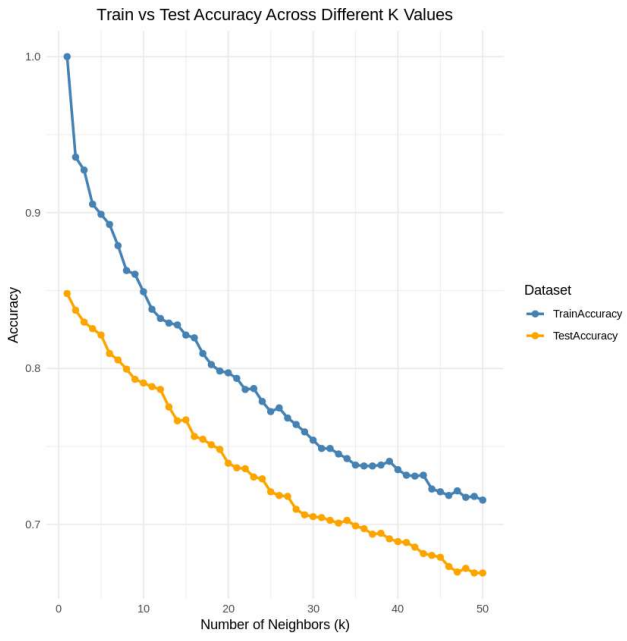
```r
# Convert factor labels to numeric (must match levels)
true_labels <- as.numeric(y_test)
pred_labels <- as.numeric(final_knn_pred)

# Calculate Mean Squared Error (MSE)
knn_mse <- mse(true_labels, pred_labels)
cat("KNN Test Set MSE:", round(knn_mse, 4), "\n")
```

```
KNN Test Set MSE: 1.4381
```

```r
# Create the confusion matrix using predicted and actual labels
conf_matrix_knn <- confusionMatrix(final_knn_pred, y_test)
# Convert the confusion matrix table to a long-format data frame
cm_df <- as.data.frame(conf_matrix_knn$table)
colnames(cm_df) <- c("True", "Predicted", "Freq")
# Plot the confusion matrix as a heatmap
ggplot(cm_df, aes(x = Predicted, y = True, fill = Freq)) +
  geom_tile(color = "white") +
  geom_text(aes(label = Freq), color = "black", size = 3) +
  scale_fill_gradient(low = "white", high = "steelblue") +
  labs(
    title = "KNN Confusion Matrix",x = "Predicted Label",y = "True Label") +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1),
    plot.title = element_text(hjust = 0.5))
```




```
# Create a data frame
accuracy_results <- data.frame(k = 1:50,TrainAccuracy = train.accuracy,TestAccuracy = cv.
accuracy)

# Reshape data to long format for plotting multiple lines
accuracy_long <- melt(
  accuracy_results,id.vars = "k",variable.name = "Dataset", value.name = "Accuracy")

# Plot Train vs Test Accuracy across different k values
ggplot(accuracy_long, aes(x = k, y = Accuracy, color = Dataset)) +
  geom_line(size = 1) +
  geom_point(size = 2) +
  labs(title = "Train vs Test Accuracy Across Different K Values",
    x = "Number of Neighbors (k)",y = "Accuracy") +
  scale_color_manual(
    values = c("TrainAccuracy" = "steelblue", "TestAccuracy" = "orange"))+
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



The K-Nearest Neighbors (KNN) model was evaluated using **5-fold cross-validation** to determine the optimal value of k. The best-performing configuration was found at **k = 1** which achieved a **cross-validation accuracy of 0.848** and a **test set accuracy of 0.8595**. These results indicate that the model was able to generalize well to unseen data while maintaining strong performance on the training data.

The **confusion matrix** demonstrates that most predictions were accurate particularly for **clearly defined categories such as Obesity_Type_I, Obesity_Type_II, and Obesity_Type_III**. However, some misclassifications occurred between adjacent or borderline classes. For instance, instances labeled Insufficient_Weight were occasionally predicted as Normal_Weight, and Overweight_Level_I was often confused with Overweight_Level_II. This suggests that KNN, while effective overall, may struggle with subtle class boundaries when feature overlap exists.

The test set **Mean Squared Error (MSE) was 1.4381** which is considered acceptable given the 7-category multiclass setting. This value suggests that most misclassifications were within 1–2 class distances which aligns with the patterns observed in the confusion matrix.

Furthermore, the **accuracy trend** across different k values revealed a **clear overfitting pattern at low k values.** Training accuracy was nearly perfect at k = 1 but decreased as k increased while test accuracy peaked early and gradually declined. This indicates that **higher k values led to underfitting**. Despite this, k = 1 offered the best trade-off between variance and bias in this dataset.

Decision Tree (Contribute: Dongni Li)

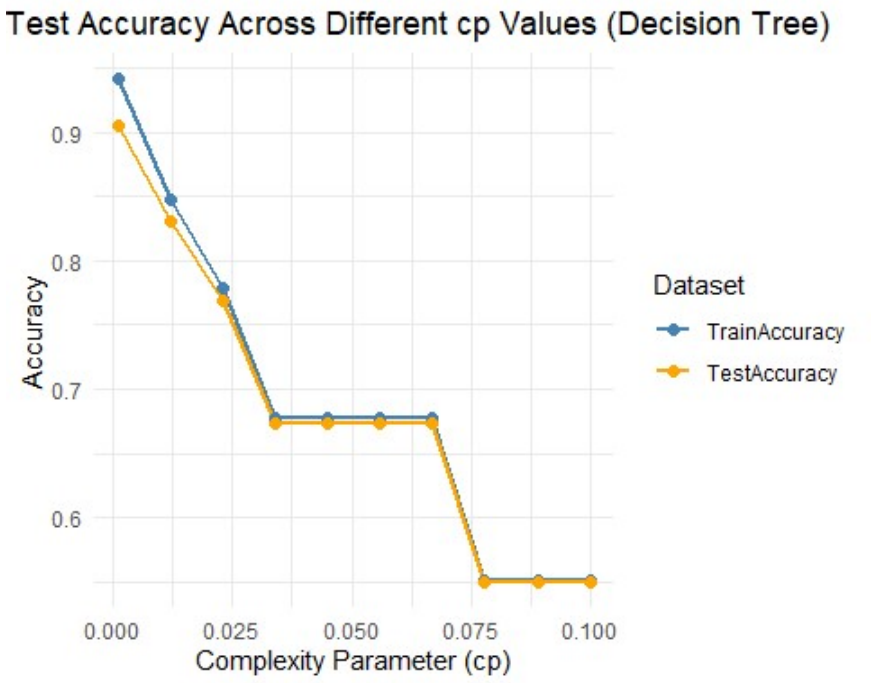
```
# Use the raw data directly (Decision Trees do not require standardization)
set.seed(123)
ctrl <- trainControl(method = "cv", number = 5,
                     summaryFunction = multiClassSummary,
                     classProbs = TRUE)
# Parameter tuning grid
tune_grid <- expand.grid(
  cp = seq(0.001, 0.1, length.out = 10)  # Complexity parameter
)
# Train the decision tree model
tree_model <- train(NObeyesdad ~ .,
                    data = train_data,
                    method = "rpart",
                    trControl = ctrl,
                    tuneGrid = tune_grid,
                    metric = "Accuracy")  # Evaluate using accuracy

# View the best parameter
print(tree_model$bestTune)

     cp
1 0.001

# Retrain decision tree using the best cp (0.001) on the full training data
final_tree <- rpart(NObeyesdad ~ ., data = train_data, control = rpart.control(cp = 0.00
1))

# Predict on the training set
train_pred <- predict(final_tree, newdata = train_data, type = "class")
train_accuracy <- mean(train_pred == train_data$NObeyesdad)
cat("Decision Tree Train Set Accuracy (cp = 0.001):", round(train_accuracy, 4), "\n")
```

```
Decision Tree Train Set Accuracy (cp = 0.001): 0.9415
```

```r
# Predict on the test set
test_pred <- predict(final_tree, newdata = test_data, type = "class")
test_accuracy <- mean(test_pred == test_data$NObeyesdad)
cat("Decision Tree Test Set Accuracy (cp = 0.001):", round(test_accuracy, 4), "\n")
```

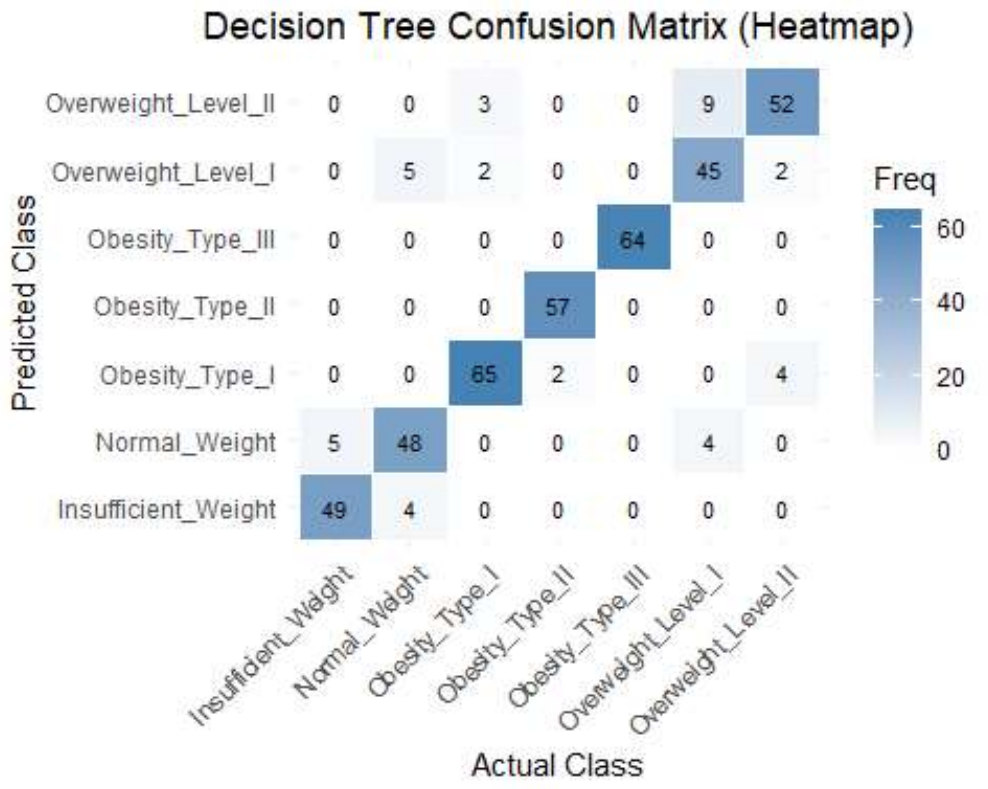
```
Decision Tree Test Set Accuracy (cp = 0.001): 0.9048
```

```r
# Define cp values to evaluate
cp_values <- seq(0.001, 0.1, length.out = 10)

# Initialize vectors to store accuracies
train_accuracies <- c()
test_accuracies <- c()

# Loop over each cp value
for (cp_val in cp_values) {
  # Train a decision tree with given cp
  tree_model_cp <- rpart(
    NObeyesdad ~ ., data = train_data, control = rpart.control(cp = cp_val))

  # Predict on train set
  train_pred <- predict(tree_model_cp, newdata = train_data, type = "class")
  train_acc <- mean(train_pred == train_data$NObeyesdad)

  # Predict on test set
  test_pred <- predict(tree_model_cp, newdata = test_data, type = "class")
  test_acc <- mean(test_pred == test_data$NObeyesdad)

  # Store results
  train_accuracies <- c(train_accuracies, train_acc)
  test_accuracies <- c(test_accuracies, test_acc)}

# Combine into a data frame
accuracy_df <- data.frame(
  cp = cp_values,
  TrainAccuracy = train_accuracies,
  TestAccuracy = test_accuracies)

# Reshape for plotting
accuracy_long <- melt(
  accuracy_df, id.vars = "cp", variable.name = "Dataset", value.name = "Accuracy")

# Plot
ggplot(accuracy_long, aes(x = cp, y = Accuracy, color = Dataset)) +
  geom_line(size = 1) +
  geom_point(size = 2) +
  labs(title = "Train vs Test Accuracy Across Different cp Values (Decision Tree)",
    x = "Complexity Parameter (cp)",y = "Accuracy") +
  scale_color_manual(
    values = c("TrainAccuracy" = "steelblue", "TestAccuracy" = "orange")) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



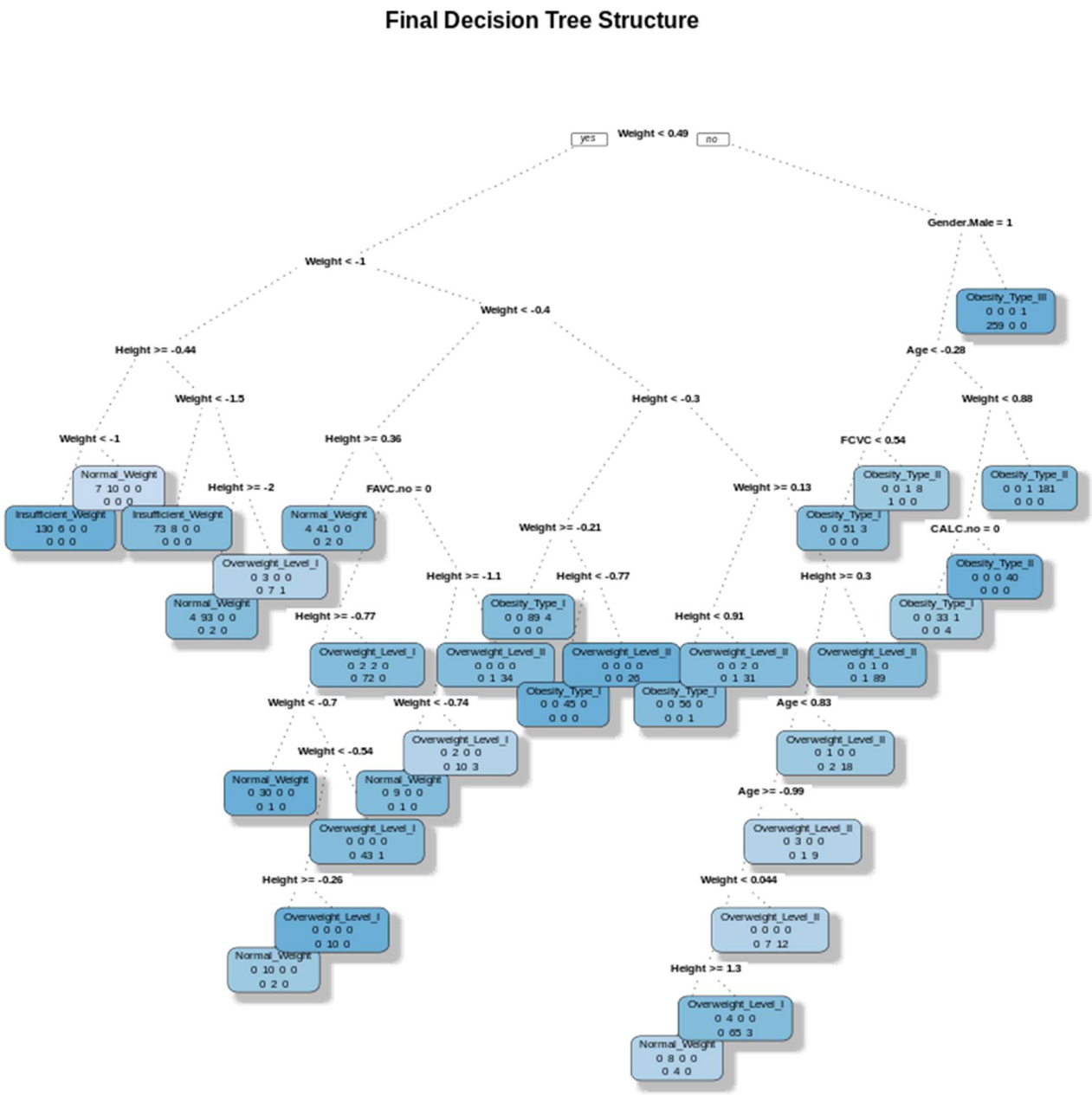
The Decision Tree classifier was trained using a range of complexity parameter (cp) values to balance model simplicity and predictive performance. The plot of train and test accuracy across different cp values demonstrates a clear trade-off between overfitting and underfitting. As cp increases, the model becomes more regularized, resulting in a decrease in both training and testing accuracy. The best performance was observed at **cp = 0.001** where the tree retained sufficient depth to capture the structure of the data without overfitting excessively. At this optimal setting, the model achieved a **training accuracy of 0.9415 and a test accuracy of 0.9048.**

```
# Predictions and evaluation
tree_pred <- predict(tree_model, newdata = test_data)

# Create the confusion matrix
conf_matrix <- confusionMatrix(tree_pred, test_data$NObeyesdad)

# Convert the confusion matrix table into a long-format data frame
cm_df <- as.data.frame(conf_matrix$table)
colnames(cm_df) <- c("Predicted", "Actual", "Freq")

# Plot the confusion matrix as a heatmap
ggplot(cm_df, aes(x = Actual, y = Predicted, fill = Freq)) +
  geom_tile(color = "white") +
  geom_text(aes(label = Freq), color = "black", size = 3) +
  scale_fill_gradient(low = "white", high = "steelblue") +
  labs(
    title = "Decision Tree Confusion Matrix (Heatmap)",
    x = "Actual Class",
    y = "Predicted Class") +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1),
    plot.title = element_text(hjust = 0.5)
  )
```

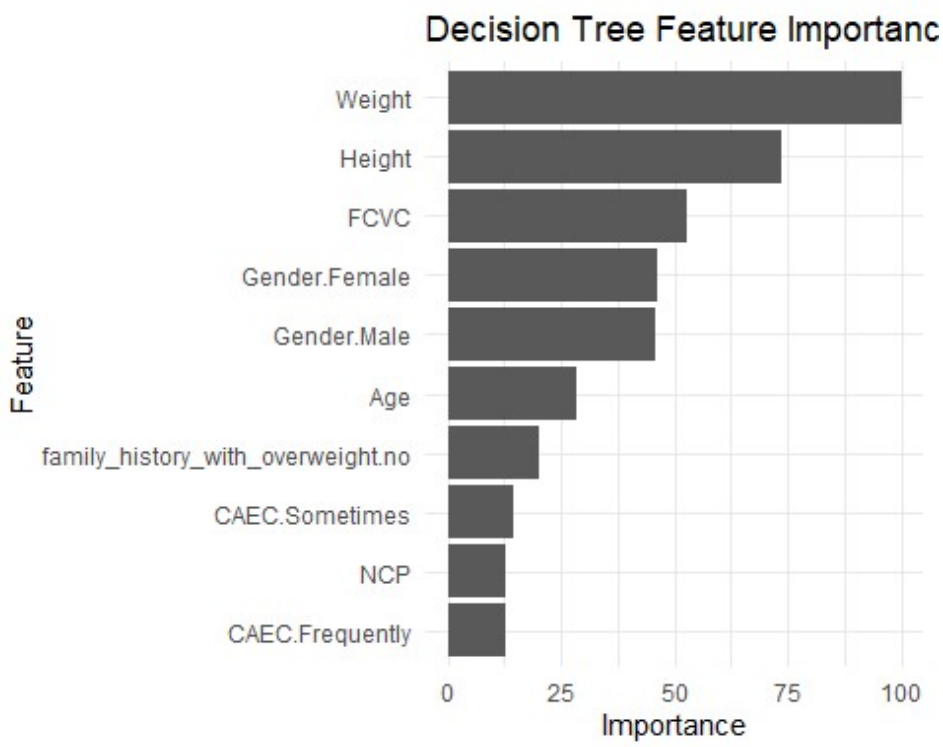


The relatively small gap between the two accuracy scores suggests a well-generalized model with limited overfitting. This conclusion is further supported by the **confusion matrix** which shows that **most classes especially Obesity_Type_I, Obesity_Type_II, and Obesity_Type_III, were predicted with high precision.** Nonetheless, the confusion matrix also reveals common misclassifications between neighboring classes such as Insufficient_Weight and Normal_Weight, or Overweight_Level_I and Overweight_Level_II. These confusions may stem from overlapping feature distributions or subtle class boundaries which are common limitations of decision tree models without ensemble methods like random forests or boosting.

```
# Visualize the final tree
prp(tree_model$finalModel,
    extra = 1,          # Display additional information
    varlen = 0,
    faclen = 0,
    cex = 0.4,
    branch.lty = 3,
    box.palette = "Blues",
    shadow.col = "gray",
    main = "Final Decision Tree Structure")
```




```r
# Variable importance
var_imp <- varImp(tree_model)
ggplot(var_imp, top = 10) +
  labs(title = "Decision Tree Feature Importance") +
  theme_minimal()
```



The plot shows the **feature importance** from the decision tree model.

- **Weight** is the **most important feature** for predicting obesity levels.
- Height and FCVC (vegetable consumption) are also significant.
- Gender (Male and Female) and Age have moderate importance.
- Features like NCP(Number of meals per day) and CAEC.Frequently (Consumption between meals) are less influential.

Support Vector Machines (SVM) (Contribute: Meng Hsuan Ho)

I first remove no information gain constant feature to avoid error. Then I employed the e1071::tune() function to search across multiple kernel functions (linear, polynomial, radial, and sigmoid) and cost values. The best-performing combination was found to be **a linear kernel** with a **cost parameter of 100**, based on **10-fold cross-validation accuracy**.

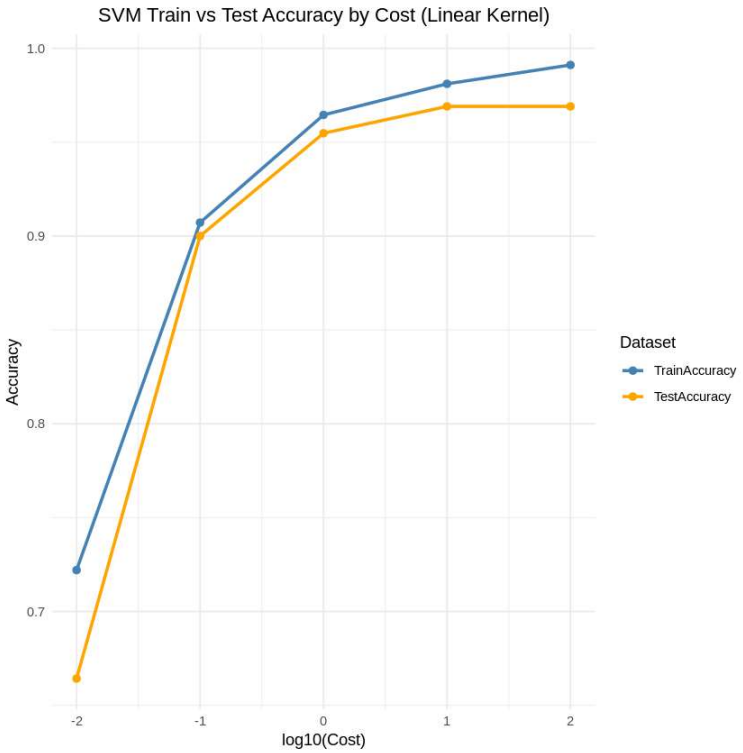
```
# Step 1: Remove constant (zero variance) features
nzv <- nearZeroVar(train_data, saveMetrics = TRUE)

# Keep only variables that are NOT constant
train_data_clean <- train_data[, !nzv$zeroVar]
test_data_clean <- test_data[, colnames(train_data_clean)]

# Make sure the target variable is still a factor
train_data_clean$NObeyesdad <- as.factor(train_data_clean$NObeyesdad)

# Step 2: SVM tuning
set.seed(123)
svm_tuned <- tune(
  svm, NObeyesdad ~ ., data = train_data_clean,
  ranges = list(
    kernel = c("linear", "polynomial", "radial", "sigmoid"),
    cost = 10^(-1:2)))
```

```r
# Print the best combination of parameters (kernel and cost)
print(svm_tuned$best.parameters)
```

```
   kernel cost
13 linear  100
```

```r
best_model_svm <- svm_tuned$best.model
```

```r
# Step 3: Evaluate model performance
# model overfitting diagnose

# Define a sequence of cost values to test
cost_values <- 10^seq(-2, 2, by = 1)  # cost = 0.01, 0.1, 1, 10, 100

# Initialize a data frame to store accuracy results
svm_accuracy_results <- data.frame(
  Cost = numeric(),TrainAccuracy = numeric(),TestAccuracy = numeric())

# Train an SVM model for each cost value
for (c in cost_values) {
  # Train the SVM model with a linear kernel
  model <- svm(NObeyesdad ~ ., data = train_data_clean, cost = c, kernel = "linear")

  # Make predictions on the training and test sets
  train_pred <- predict(model, newdata = train_data_clean)
  test_pred  <- predict(model, newdata = test_data_clean)

  # Compute accuracy for both sets
  train_acc <- mean(train_pred == train_data_clean$NObeyesdad)
  test_acc  <- mean(test_pred == test_data_clean$NObeyesdad)

  # Append the results to the accuracy data frame
  svm_accuracy_results <- rbind(svm_accuracy_results, data.frame(
    Cost = c,TrainAccuracy = train_acc,TestAccuracy = test_acc))}

# Reshape data to long format for ggplot visualization
svm_accuracy_long <- melt(
  svm_accuracy_results, id.vars = "Cost",
  variable.name = "Dataset", value.name = "Accuracy")

# Plot Train vs Test Accuracy across different cost values
ggplot(svm_accuracy_long, aes(x = log10(Cost), y = Accuracy, color = Dataset)) +
  geom_line(linewidth = 1) +
  geom_point(size = 2) +
  labs(
    title = "SVM Train vs Test Accuracy by Cost (Linear Kernel)",
    x = "log10(Cost)",
    y = "Accuracy") +
  scale_color_manual(
    values = c("TrainAccuracy" = "steelblue", "TestAccuracy" = "orange")) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



Overfitting Analysis for SVM (Linear Kernel)

The SVM model begins to show signs of **overfitting at higher cost values** (particularly at Cost = 100). When Cost = 100, training accuracy becomes nearly perfect but test accuracy does not improve correspondingly. A moderate cost value (Cost = 10) appears to offer a good trade-off between bias and variance, providing high accuracy while maintaining generalization.

```
# Predict on the test set
svm_pred <- predict(best_model_svm, newdata = test_data_clean)

# Accuracy
accuracy <- mean(svm_pred == test_data_clean$NObeyesdad)
cat("SVM Test Accuracy (Cost = 10):", round(accuracy, 4), "\n")
```

```
SVM Test Accuracy (Cost = 10): 0.969
```

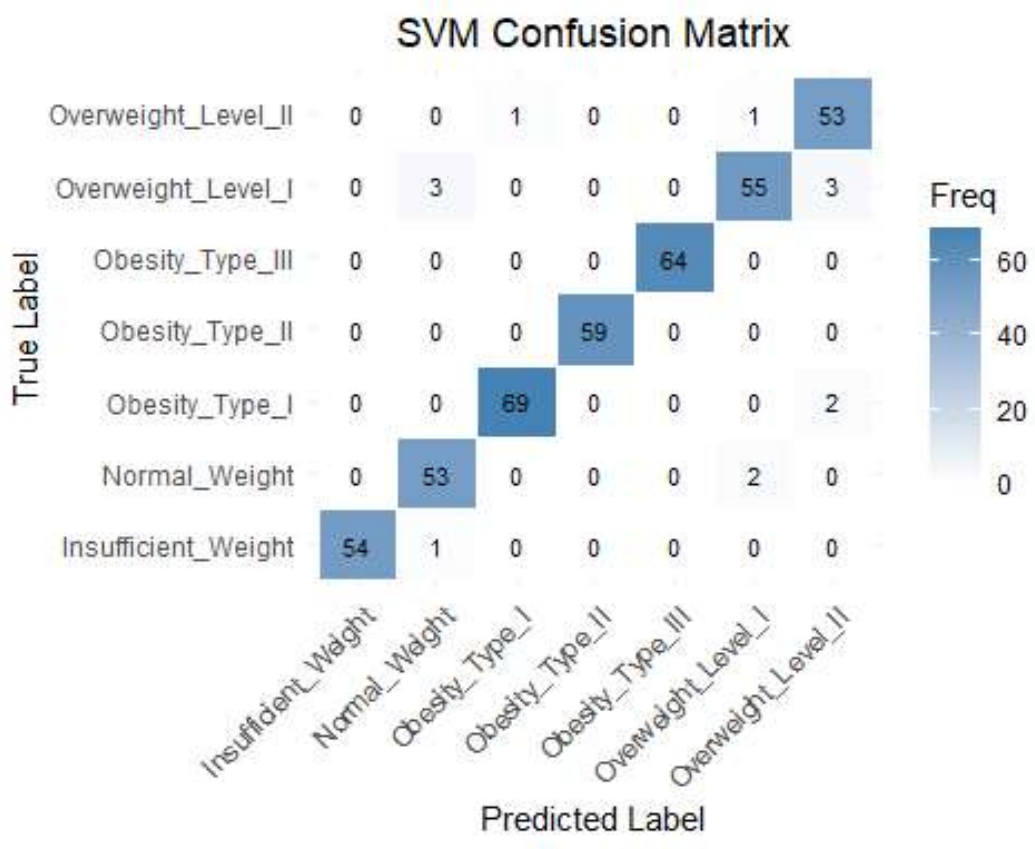
```
# Mean Squared Error (MSE)
true_labels <- as.numeric(test_data_clean$NObeyesdad)
pred_labels <- as.numeric(svm_pred)
mse_val <- mse(true_labels, pred_labels)
cat("SVM Mean Squared Error (Cost = 10):", round(mse_val, 4), "\n")
```

```
SVM Mean Squared Error (Cost = 10): 0.3167
```

Accuracy & Mean Squared Error (MSE)

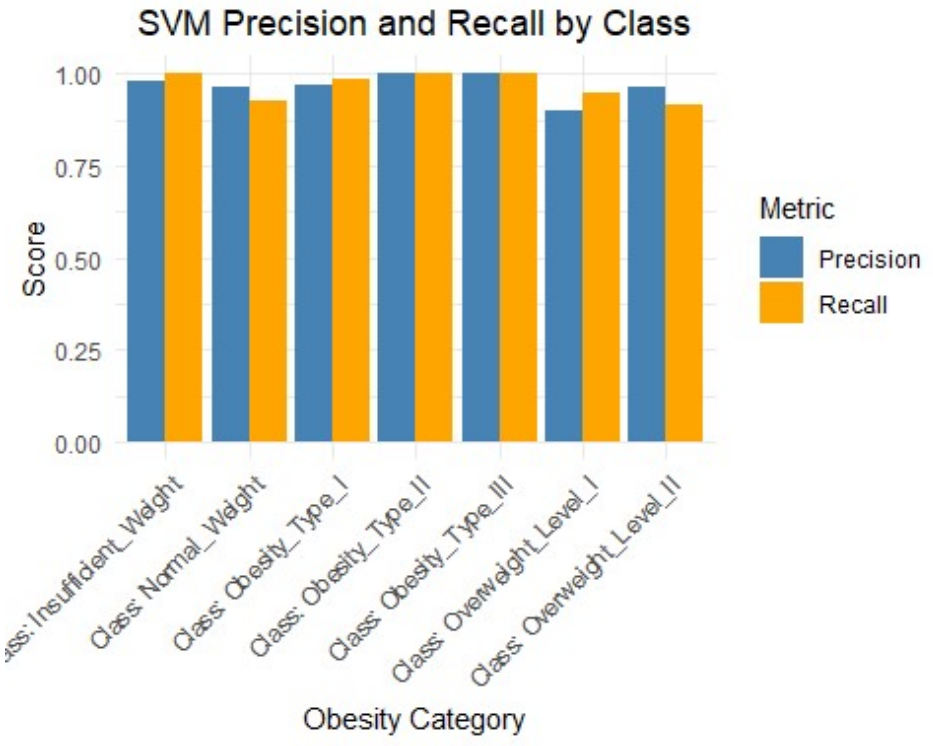
Based on the comparative evaluation of multiple classification algorithms, the Support Vector Machine (SVM) with a **linear kernel and cost = 100** emerged as the best-performing model for predicting obesity levels. This conclusion is supported by the model's outstanding performance metrics on the **test set** including an **accuracy of 0.969** and a **mean squared error (MSE) of 0.3167**.

```r
# Confusion Matrix
conf_matrix_svm <- confusionMatrix(svm_pred, test_data_clean$NObeyesdad)

# Convert the confusion matrix table to a long-format data frame
cm_svm_table <- as.table(conf_matrix_svm$table)
cm_svm_df <- as.data.frame(cm_svm_table)
colnames(cm_svm_df) <- c("True", "Predicted", "Freq")

# Create the heatmap plot
ggplot(cm_svm_df, aes(x = Predicted, y = True, fill = Freq)) +
  geom_tile(color = "white") +
  geom_text(aes(label = Freq), color = "black", size = 3) +
  scale_fill_gradient(low = "white", high = "steelblue") +
  labs(
    title = "SVM Confusion Matrix",
    x = "Predicted Label",
    y = "True Label") +
  theme_minimal() +  # Clean minimal theme
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1),
    plot.title = element_text(hjust = 0.5))
```



The **SVM confusion matrix** demonstrates that most predictions fall along the diagonal, suggesting that the **SVM model accurately identified instances across all obesity categories**. Minor misclassifications occurred primarily between adjacent categories such as Overweight_Level_I and Overweight_Level_II or between Insufficient_Weight and Normal_Weight. The misclassifications were expected due to the natural overlap between neighboring weight categories.

```r
# Extract per-class metrics from the confusion matrix
metrics_svm <- as.data.frame(conf_matrix_svm$byClass)

# If this is a multi-class classification, each row corresponds to a class
# Extract Precision and Recall columns
metrics_svm_df <- data.frame(Class = rownames(metrics_svm),Precision = metrics_svm$Precis
ion,Recall = metrics_svm$Recall)

# Reshape data to long format for grouped bar chart
metrics_svm_long <- melt(metrics_svm_df, id.vars = "Class", variable.name = "Metric", val
ue.name = "Value")

# Plot bar chart of Precision and Recall for each class
ggplot(metrics_svm_long, aes(x = Class, y = Value, fill = Metric)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "SVM Precision and Recall by Class", x = "Obesity Category",y = "Score") +
  scale_fill_manual(values = c("steelblue", "orange")) +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1),
    plot.title = element_text(hjust = 0.5))
```

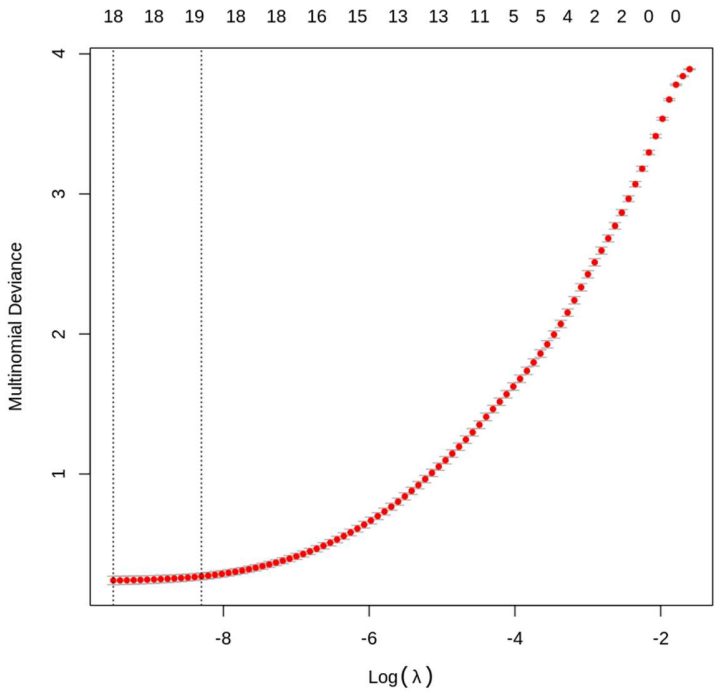


To further evaluate the model's robustness, we examined SVM per-class precision and recall scores. **Most classes achieved precision and recall above 0.95** with particularly high scores for Obesity_Type_II and Obesity_Type_III. This indicates that the model performed well on these distinct classes. Slightly lower recall was observed for Normal_Weight and Overweight_Level_II, reflecting some difficulty in distinguishing borderline cases.

Overall, when properly tuned the SVM model, demonstrated strong generalization ability and effectively distinguished between multiple obesity levels. It is makinga suitable choice for this multiclass classification task.

Logistic Regression with Lasso (Contribute: Dongni Li)

```
# use model.matrix() to properly handle categorical variables
x_train <- model.matrix(NObeyesdad ~ ., data = train_data)[, -1]
x_test <- model.matrix(NObeyesdad ~ ., data = test_data)[, -1]

y_train <- train_data$NObeyesdad
y_test <- test_data$NObeyesdad

# Train the Lasso model (with default alpha = 1, which represents Lasso)
lasso_model <- glmnet(x_train, y_train, alpha = 1, family = "multinomial")

# Perform cross-validation to select the best lambda (regularization parameter)
cv_lasso <- cv.glmnet(x_train, y_train, alpha = 1, family = "multinomial")

# Plot the cross-validation results to visualize the best lambda value
plot(cv_lasso)  # Plotting lambda values against the cross-validation error
```



The plot shows how the multinomial deviance changes with the log of lambda (regularization parameter) during cross-validation in the Lasso model. The vertical dotted lines mark the optimal lambda values, where the deviance is minimized. The plot shows that **smaller lambda values result in better fits,** but after a certain point, larger lambda leads to higher deviance (worse fit).

```
# Display the best lambda selected through cross-validation
best_lambda <- cv_lasso$lambda.min
cat("Lasso Best lambda:", best_lambda, "\n")
```

```
Lasso Best lambda: 7.403543e-05
```

```
# Make predictions using the best lambda value
# Predict on the training set using the optimal lambda
# Use type = "class" to obtain predicted class labels
train_pred <- predict(cv_lasso, newx = x_train, s = "lambda.min", type = "class")
```

```r
train_accuracy <- mean(train_pred == y_train)
cat("Lasso Train Accuracy:", round(train_accuracy, 4), "\n")
```

```
Lasso Train Accuracy: 0.9888
```

```r
# Predict on the test set using the optimal lambda
test_pred <- predict(cv_lasso, newx = x_test, s = "lambda.min", type = "class")
test_accuracy <- mean(test_pred == y_test)
cat("Lasso Test Accuracy:", round(test_accuracy, 4), "\n")
```

```
Lasso Test Accuracy: 0.9643
```

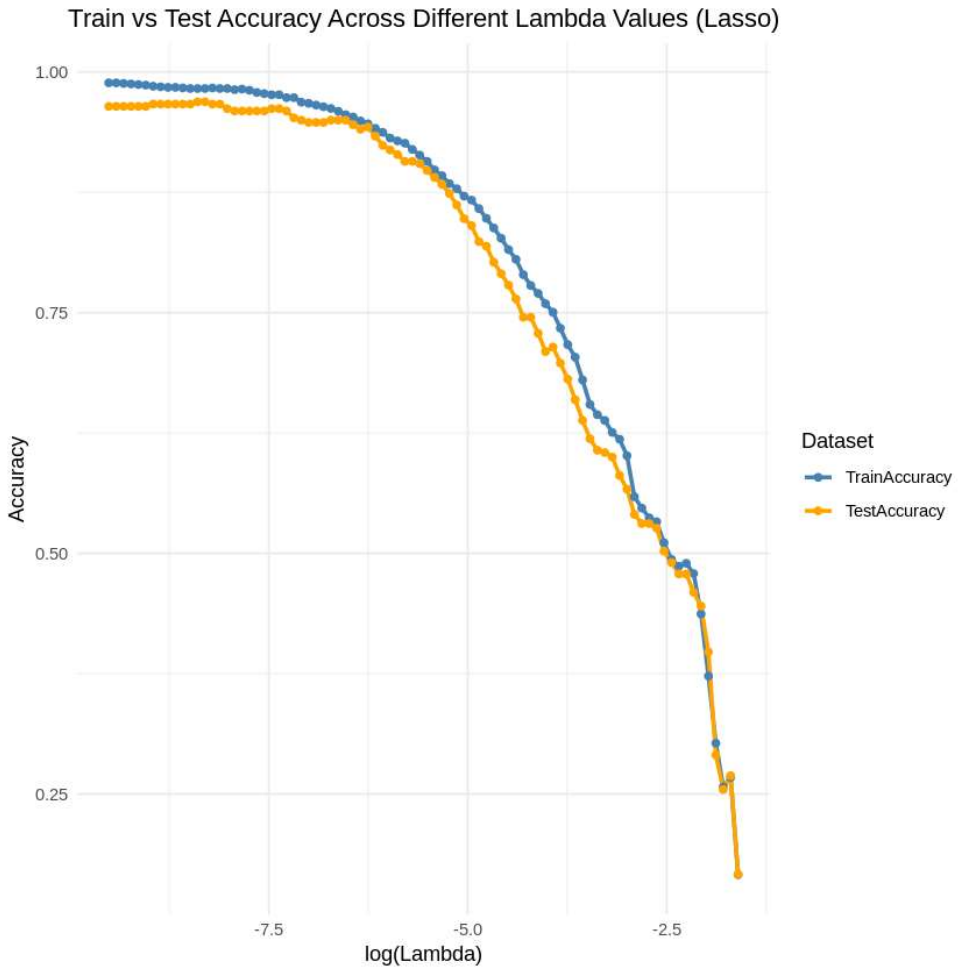
```r
# Make sure both are factors first
true_labels <- as.numeric(factor(y_test))
pred_labels <- as.numeric(factor(test_pred, levels = levels(y_test)))  # levels aligned w
ith y_test

# Now safely calculate MSE
lasso_mse <- mse(true_labels, pred_labels)
cat("Lasso Test Set MSE:", round(lasso_mse, 4), "\n")
```

```
Lasso Test Set MSE: 0.2214
```

The Lasso model used L1 regularization to reduce overfitting and improve generalization. Using 5-fold cross-validation, the **best lambda** was selected as 7.40×10^{-5}. At this setting, the model achieved **98.88% accuracy on the training set**, **96.43% accuracy on the test set**, and **22.14% MSE rate on the test set**.

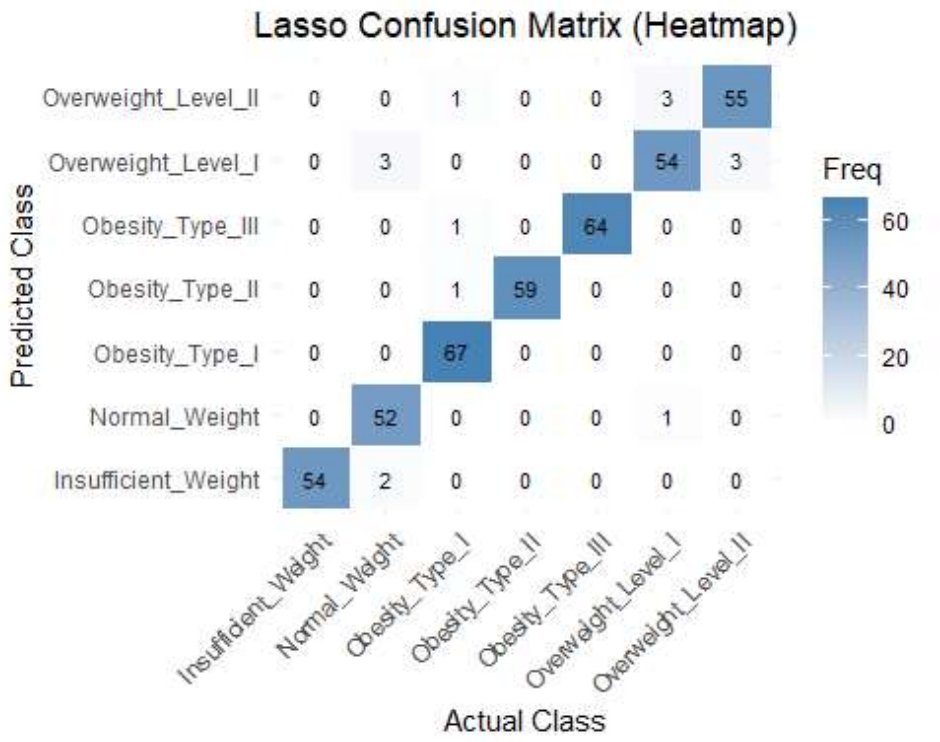
```r
# Extract lambda sequence from cv.lasso
lambda_seq <- cv_lasso$lambda

# Initialize vectors
train_accuracies <- c()
test_accuracies <- c()

# Loop through each lambda value
for (lambda_val in lambda_seq) {
  # Predict on train set
  train_pred <- predict(cv_lasso, newx = x_train, s = lambda_val, type = "class")
  train_acc <- mean(train_pred == y_train)

  # Predict on test set
  test_pred <- predict(cv_lasso, newx = x_test, s = lambda_val, type = "class")
  test_acc <- mean(test_pred == y_test)

  # Store the results
  train_accuracies <- c(train_accuracies, train_acc)
  test_accuracies <- c(test_accuracies, test_acc)
}

# Combine into a data frame
accuracy_df <- data.frame(
  Lambda = lambda_seq,
  TrainAccuracy = train_accuracies,
  TestAccuracy = test_accuracies
)
```

```r
# Reshape data for plotting
accuracy_long <- melt(
  accuracy_df, id.vars = "Lambda",
  variable.name = "Dataset", value.name = "Accuracy")

# Plot Train vs Test Accuracy
ggplot(accuracy_long, aes(x = log(Lambda), y = Accuracy, color = Dataset)) +
  geom_line(size = 1) +
  geom_point(size = 1.5) +
  labs(
    title = "Train vs Test Accuracy Across Different Lambda Values (Lasso)",
    x = "log(Lambda)",
    y = "Accuracy") +
  scale_color_manual(
    values = c("TrainAccuracy" = "steelblue", "TestAccuracy" = "orange")) +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



The small gap between training and testing accuracies suggests **minimal overfitting**. Additionally, the Train vs Test Accuracy Curve across different lambda values demonstrates a classical bias-variance trade-off: low lambda values yield high training accuracy but risk overfitting, while high lambda values simplify the model excessively, leading to underfitting. The selected lambda clearly lies in the sweet spot of this curve.

```r
# Ensure both predicted and actual labels are factors
predicted_factor <- factor(test_pred, levels = levels(y_test))
actual_factor <- factor(y_test)

# Create confusion matrix
conf_matrix <- confusionMatrix(predicted_factor, actual_factor)

# Convert confusion matrix table into a long-format data frame
cm_df <- as.data.frame(conf_matrix$table)
colnames(cm_df) <- c("Predicted", "Actual", "Freq")

# Plot the confusion matrix as a heatmap
ggplot(cm_df, aes(x = Actual, y = Predicted, fill = Freq)) +
  geom_tile(color = "white") +
  geom_text(aes(label = Freq), color = "black", size = 3) +
  scale_fill_gradient(low = "white", high = "steelblue") +
  labs(
    title = "Lasso Confusion Matrix (Heatmap)",
    x = "Actual Class",
    y = "Predicted Class") +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1),
    plot.title = element_text(hjust = 0.5)
  )
```



The confusion matrix further validates the model's robustness. **Most classes were predicted accurately** including challenging adjacent categories such as Overweight_Level_I vs Overweight_Level_II and Insufficient_Weight vs Normal_Weight. Only a few minor misclassifications were observed which confirms the model's discriminative ability across nuanced class boundaries.

Conclusion (Contribute: Meng Hsuan Ho)

Based on the comparative evaluation of five modeling approaches: Linear Discriminant Analysis (LDA), K-Nearest Neighbors (KNN), Decision Trees, Support Vector Machines (SVM), and Logistic Regression with Lasso regularization, the **two top-performing models were Lasso and SVM**.

The **Lasso model** achieved a **test accuracy of 96.43%** and a **test mean squared error (MSE) of 0.2214**, demonstrating strong predictive power and excellent generalization with minimal overfitting.

The **SVM model** achieved a slightly higher **test accuracy of 96.90%**, but its **MSE was slightly higher at 0.3167** compared to Lasso.

While both models showed outstanding performance, the Lasso model's slightly lower MSE suggests more stable predictions particularly for closely related obesity categories.

In contrast, the LDA, KNN, and Decision Tree models achieved lower test accuracies and exhibited either greater variability across folds or more frequent misclassifications between neighboring classes.

Considering both predictive accuracy and stability, the **Lasso model is selected as the most suitable model** for obesity level classification in this study. Its combination of high accuracy, robust generalization, and effective feature regularization makes it the best choice for practical deployment.