Comparative Analysis of KNN and SVM Models in Health Outcomes of Horses

Authors: Armando, Derick, Rachel, Cong, Lawrence

Student IDs:

I. Introduction

Background: This project is centered on the predictive analysis of health outcomes in horses, an area of growing interest in veterinary science and horse racing. Using advanced statistical models, we aim to predict which horses are in the best condition to race. Our focus is on implementing and comparing the K-Nearest Neighbors (KNN) and Support Vector Machine (SVM) models to achieve this goal.

Research Question: Our primary objective is to determine the factors that most significantly influence the health outcomes of horses and predict these outcomes effectively. Specifically, we aim to answer: "Which horses are in the best condition to race based on various health indicators?"

Dataset Description: The dataset for this analysis, titled 'Horse', is sourced from Walter Reade and Ashley Chow (2023) on Kaggle under the competition "Predict Health Outcomes of Horses" (Predict Health Outcomes of Horses | Kaggle). It comprises data on various health parameters and outcomes for horses, offering a comprehensive view of their medical status.

- **Dataset**: Our dataset consists of 1235 rows and 29 columns, encompassing a range of variables from surgical history to detailed physiological measurements.

The data frame includes crucial variables like surgery status, age, hospital number, rectal temperature, pulse, respiratory rate, and many more, each contributing uniquely to the overall health assessment of the horses. Notably, the 'outcome' variable, indicating whether a horse died, was euthanized, or survived, serves as our primary response variable for model prediction.

II. Methodology

KNN Model (Armando, Derick, Rachel)

Overview of KNN Model: The K-nearest Neighbors (KNN) algorithm is a popular and intuitive machine learning approach used for both classification and regression tasks, preferably for classification as it assumes observations close to each other are similar. KNN is non-parametric and instance-based, making predictions based on the majority class or average of the k-nearest data points in the dataset. In our horse health outcome project, KNN emerges as a suitable candidate for its simplicity and ease of implementation. Given our dataset's dimensionality of 27 predictors, KNN's ability to handle multi-dimensional data without assuming a specific functional form aligns well with our objectives. A disadvantage that could hinder the model would be that KNN tends to run into overfitting. This means the function would be too closely aligned to a limited set of data points. Additionally, KNN does not scale well, meaning it will utilize more memory and data than other classifiers.

Model Selection and Leave-One-Out Cross Validation:

- **Validation Set Approach**: The most popular approach method that splits the data accordingly and performs KNN on the data.
- **Leave-One-Out Cross Validation**: Utilized to avoid inconsistencies such as high variability depending on the train/test data split.

Model Training and Validation:

- We split the data into training (80%) and testing (20%) sets, maintaining the distribution of outcomes across both sets.
- The KNN model was trained on the training set with (placeholder)
- Predictions were made on the test set.

Model Formula: The prediction for a given observation in KNN is determined by identifying the K points in the training set that are closest to a set of points based on the majority or average of these neighbors.

- **KNN Model Formula**: The KNN model formula uses euclidean distance to calculate the distance between one point and another. ie: d(x,y) = $\sqrt{\sum_{i=1}^{n} (x_a - y_i)^2}$

SVM Model (Cong, Lawrence)

Overview of SVM Model: The Support Vector Machine (SVM) is a powerful and versatile machine learning model used for classification tasks. SVM works by finding the optimal hyperplane that best separates different classes in a high-dimensional space. In our project, we want to classify the health outcomes of horses based on various. physiological and medical parameters. And since our dataset has up to 27 predictors, SVM is the right candidate for this task. Now, one disadvantage that SVM has that we foresee is that it is sensitive to noise and outliers, affecting the position of the decision boundary. Nevertheless, with preprocessing and normalization, it will be greatly minimized. Additionally, SVM is very resource intensive, limiting our capabilities in finding the right parameters as we would see later in the hyperparameter tuning section of the report.

Model Selection and Parameter Tuning:

- **Kernel Selection**: We experimented with different kernel functions (linear, polynomial, and radial basis function (RBF)) to determine which best fits our data.
- **Hyperparameter Tuning**: Critical parameters like the regularization parameter (C) and the kernel-specific parameters (like gamma in the RBF kernel) were fine-tuned. We used grid search with cross-validation to systematically explore a wide range of parameter combinations and identify the optimal settings.

Model Training and Validation:

- We split the data into training (80%) and testing (20%) sets as we did in the KNN method, maintaining the distribution of outcomes across both sets.
- The SVM model was trained on the training set with the selected kernel and parameters.
- Model performance was evaluated on the test set using metrics like training, testing accuracy and training error.

Model Formula:

- **Linear SVM:** For linear SVM, the decision function is a linear combination of the input features: $f(x)=w \cdot x+b$, where w is the weight vector, x is the feature vector, and b is the bias.
- **RBF Kernel**: For the RBF kernel, the decision function is based on the radial basis function: $K(x,xi)=\exp(-\gamma||x - xi||^2)$, where x and xi are feature vectors, and γ is the kernel coefficient.
- **Polynomial Kernel:** For the polynomial kernel, the decision function is $K(x,xi)=(\alpha x \cdot xi+r)^d$, where dd is the degree of the polynomial, α is the scale parameter, and r is the independent term.

III. Data Analysis

KNN Model Analysis (Armando, Derick, Rachel)

Preprocessing and Feature Selection: We ensured that there were no missing values in the data set. We normalized the data set since KNN is deeply affected by these values (The use of euclidean distance causes this). All categorical variables were converted to numerical.

Model Fitting and Tuning: Our analysis involved splitting the data into two sets (A training and a validation set). For the model tuning part, we played around with the K value in KNN. We tried different K values - 1, 5, and 10 - to see which one gives us the best results (accuracy). This helped us pick the best value of K. which in our case was k = 5.

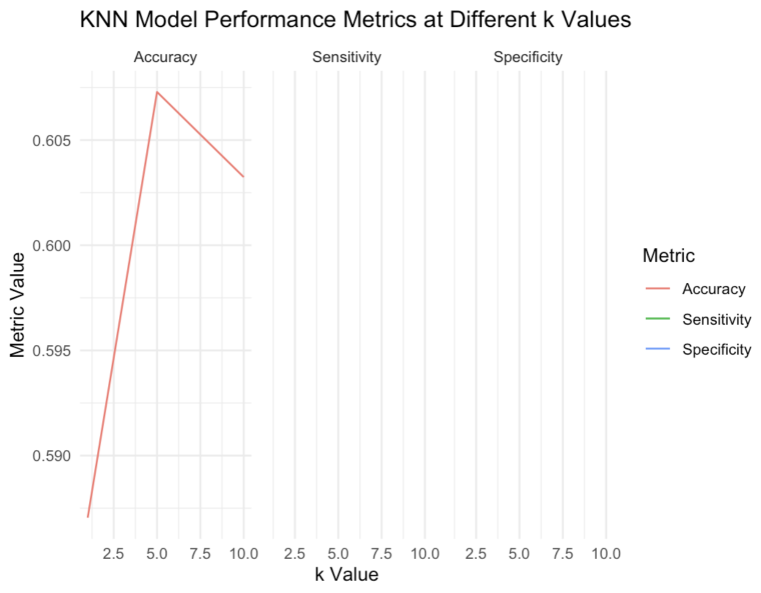
Results and Interpretation:

KNN Model with k = 1

Accuracy: 58.7%. This suggests that when using only the nearest neighbor for each prediction, the model correctly predicts the outcome about 58.7% of the time.
Sensitivity/Recall: Varies across the classes (52.94% for Class 1, 43.55% for Class 2, and 70.09% for Class 3). This means the model is best at detecting Class 3 and least effective at detecting Class 2.

KNN Model with k = 5
Accuracy: Improved to 60.73%. This increase suggests that considering more neighbors (5 in this case) provides a more accurate prediction.
Sensitivity/Recall: Shows a slight improvement in Class 1 and 3, but a decrease in Class 2.

KNN Model with k = 10
Accuracy: Slightly decreased to 60.32%. This suggests that increasing neighbors to 10 slightly reduces the model's accuracy.
Sensitivity/Recall: A decrease in sensitivity for Class 2 is observed.



SVM Model Analysis (Cong, Lawrence)

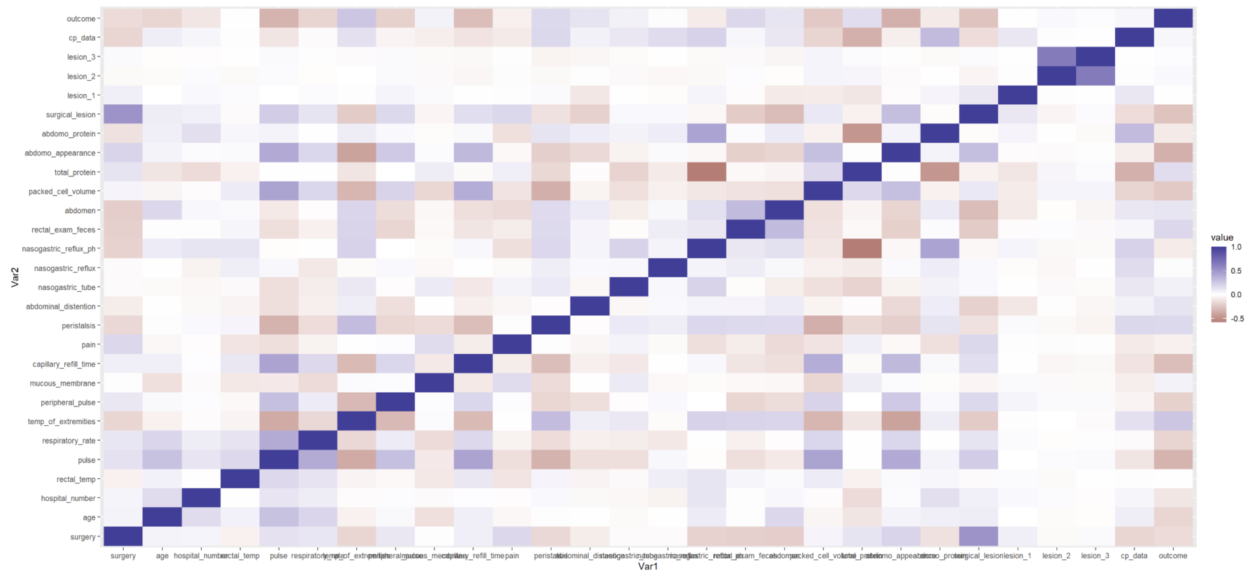


Figure 1: Correlation Matrix between Variables

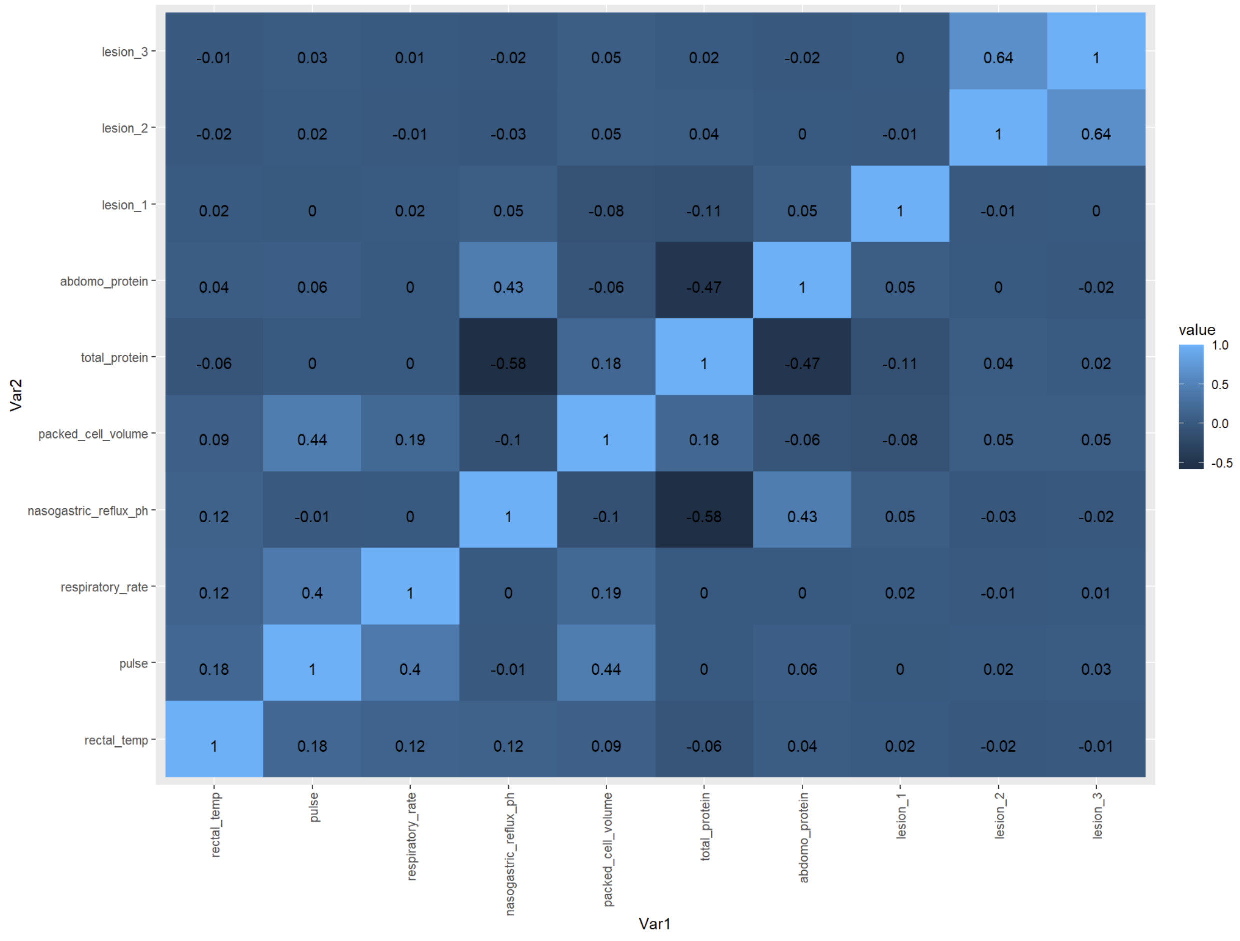


Figure 2: Correlation Matrix between Continuous Variables

Data Preprocessing: Before applying the SVM model, we conducted thorough data preprocessing, which included:

- **Handling Missing Values**: Given the complexity of the dataset, we addressed missing values appropriately, either by imputation or exclusion, depending on their impact on the model. However, since the dataset is exceedingly clean, we do not need to impute or exclude any entries.
- **Predictors Encoding**: Categorical variables, such as 'surgery', 'age', and 'temp_of_extremities', were encoded into numerical formats suitable for the SVM model.
- **Normalization**: We normalized the continuous variables like 'pulse', 'rectal_temp', and 'respiratory_rate' to ensure that each feature contributes proportionally to the model's performance. This is quite important due to the nature of SVM. SVM works by constructing a hyperplane to separate different classes, and the position of the hyperplane is influenced by the scale of the features. If one feature has a much larger range of values than others, it can dominate the objective function of the SVM and may lead to a biased hyperplane, negatively affecting the model's performance. In this case, I am using standardization or Z-score Normalization where data has the mean of 0 and a standard deviation of 1.
- **Removing Irrelevant Predictors And Multicollinearity:** Since the dataset includes 'id', we decide to drop this particular column due to the fact that it is just an identifier for each entry and doesn't correlate with the outcome or other predictive features. In addition, correlations between 'rectal_temp' and 'lesion_1' with 'outcome' are approximately 0.0055 and 0.0027 respectively, which are very low, suggesting they may not be strong predictors for the outcome. In addition, we find that correlation coefficient between 'lesion_2' and 'lesion_3' is approximately 0.644, indicating a moderate to high degree of multicollinearity as shown in Figure 1. As a result, I am dropping 'lesion_3' because in the context of the dataset, 'lesion_3' is a subtype of 'lesion_2.' Moreover, looking at Figure 2, we can see that 'total_protein' is highly correlated with 'nasogastic_reflux_ph' with the value of -0.58. Nevertheless, we decide to leave them as they are because in the context of our dataset, these two predictors depict two separate medical information.

Hyperparameter Tuning and Model Training: Our analysis involved hyperparameter tuning for SVM models with linear, radial basis function (RBF), and polynomial kernels. The tuning process focused on identifying optimal values for cost (and gamma or degree where applicable).

Cross-Validation in Model Tuning: In our SVM model tuning process, we employed 10-fold cross-validation. This method involves partitioning the training data into 10 subsets. During the tuning phase, the model was trained on 9 subsets and validated on the 1 remaining subset. This process was repeated 10 times, with each subset being used once as the validation data. This approach ensured that our model tuning was robust and that the selected hyperparameters were not just specific to a particular subset of the training data. The best-performing model parameters were determined based on their average performance across all folds.

Model Evaluation: The performance of each SVM model was assessed on both training and testing datasets, with a focus on accuracy and confusion matrices and training error.

1. **Linear Kernel**:
    a. Best Cost Parameter: 1
    b. Training Error: 0.2995259
    c. Training Accuracy: 73.48%, Testing Accuracy: 62.75%
    d. Confusion Matrix (Test Set):
        i. **For Class 1:**
            1. True Positives (TP1): 48 (correctly predicted as Class 1)
            2. False Negatives (FN1): 2 (Class 1 predicted as Class 2) + 18 (Class 1 predicted as Class 3) = 20
            3. False Positives (FP1): 20 (Class 2 predicted as Class 1) + 18 (Class 3 predicted as Class 1) = 38
            4. True Negatives (TN1): 21 + 13 + 21 + 86 = 141 (all non-Class 1 instances correctly identified)
        ii. **For Class 2:**
            1. True Positives (TP2): 21 (correctly predicted as Class 2)
            2. False Negatives (FN2): 20 (Class 2 predicted as Class 1) + 21 (Class 2 predicted as Class 3) = 41
            3. False Positives (FP2): 2 (Class 1 predicted as Class 2) + 13 (Class 3 predicted as Class 2) = 15
            4. True Negatives (TN2): 48 + 18 + 18 + 86 = 170 (all non-Class 2 instances correctly identified)
        iii. **For Class 3:**
            1. True Positives (TP3): 86 (correctly predicted as Class 3)
            2. False Negatives (FN3): 18 (Class 3 predicted as Class 1) + 13 (Class 3 predicted as Class 2) = 31
            3. False Positives (FP3): 18 (Class 1 predicted as Class 3) + 21 (Class 2 predicted as Class 3) = 39
            4. True Negatives (TN3): 48 + 2 + 20 + 21 = 91 (all non-Class 3 instances correctly identified)
2. **Radial Basis Function Kernel**:
    a. Best Cost Parameter: 1, Best Gamma: 0.1
    b. Training Accuracy: 88.36%, Testing Accuracy: 62.75%
    c. Confusion Matrix (Test Set):
        i. **For Class 1:**
            1. True Positives (TP1): 42 (correctly predicted as Class 1)
            2. False Negatives (FN1): 3 (Class 1 predicted as Class 2) + 23 (Class 1 predicted as Class 3) = 26
            3. False Positives (FP1): 14 (Class 2 predicted as Class 1) + 17 (Class 3 predicted as Class 1) = 31
            4. True Negatives (TN1): 26 + 13 + 22 + 87 = 148 (all non-Class 1 instances correctly identified)
        ii. **For Class 2:**
            1. True Positives (TP2): 26 (correctly predicted as Class 2)

2. False Negatives (FN2): 14 (Class 2 predicted as Class 1) + 22 (Class 2 predicted as Class 3) = 36
3. False Positives (FP2): 3 (Class 1 predicted as Class 2) + 13 (Class 3 predicted as Class 2) = 16
4. True Negatives (TN2): 42 + 17 + 23 + 87 = 169 (all non-Class 2 instances correctly identified)

iii. For Class 3:

1. True Positives (TP3): 87 (correctly predicted as Class 3)
2. False Negatives (FN3): 17 (Class 3 predicted as Class 1) + 13 (Class 3 predicted as Class 2) = 30
3. False Positives (FP3): 23 (Class 1 predicted as Class 3) + 22 (Class 2 predicted as Class 3) = 45
4. True Negatives (TN3): 42 + 3 + 14 + 26 = 85 (all non-Class 3 instances correctly identified)

3. **Polynomial Kernel**:
   a. Best Cost Parameter: 9, Best Degree: 1
   b. Training Error: 0.2944857
   c. Training Accuracy: 73.89%, Testing Accuracy: 63.97%
   d. Confusion Matrix (Test Set):

i. For Class 1:

1. True Positives (TP1): 48 (correctly predicted as Class 1)
2. False Negatives (FN1): 2 (Class 1 predicted as Class 2) + 18 (Class 1 predicted as Class 3) = 20
3. False Positives (FP1): 18 (Class 2 predicted as Class 1) + 18 (Class 3 predicted as Class 1) = 36
4. True Negatives (TN1): 23 + 13 + 21 + 86 = 143 (all non-Class 1 instances correctly identified)

ii. For Class 2:

1. True Positives (TP2): 23 (correctly predicted as Class 2)
2. False Negatives (FN2): 18 (Class 2 predicted as Class 1) + 21 (Class 2 predicted as Class 3) = 39
3. False Positives (FP2): 2 (Class 1 predicted as Class 2) + 13 (Class 3 predicted as Class 2) = 15
4. True Negatives (TN2): 48 + 18 + 18 + 86 = 170 (all non-Class 2 instances correctly identified)

iii. For Class 3:

1. True Positives (TP3): 86 (correctly predicted as Class 3)
2. False Negatives (FN3): 18 (Class 3 predicted as Class 1) + 13 (Class 3 predicted as Class 2) = 31
3. False Positives (FP3): 18 (Class 1 predicted as Class 3) + 21 (Class 2 predicted as Class 3) = 39
4. True Negatives (TN3): 48 + 2 + 18 + 23 = 91 (all non-Class 3 instances correctly identified)

Comparison of Models (Lawrence)

Performance Comparison:

According to the confusion matrices of the SVM models, it seems that there tends to be more false predictions for Class 3 across all methods compared to prediction of other classes. In the KNN models, Class 1 tends to be harder to predict and are therefore more likely to be misclassified.

The training accuracy of the varying SVM models average to around 78.57%, with the radial method producing the highest accuracy of 88.36%. Compared to the KNN models, the average accuracy is around 59.92%, with the model using a K-value of 5 producing the highest accuracy of 60.73%. This is lower than the training accuracy of the SVM models which had a lower average of 63.16% compared to their testing accuracies. Overall, the performance of SVM models outperform the KNN models.

Conclusion

Findings Summary: [Summarize the key findings from both models.]

SVM:

- The SVM models demonstrated varied performance across different kernels with their respective tuning. The linear kernel showed a balance between training and testing accuracy, while the RBF kernel excelled in training performance. The polynomial kernel offered a slightly higher testing accuracy. The study's approach to data preprocessing, choice of kernels, and meticulous hyperparameter tuning provided comprehensive insights into the classification of horse health outcomes using SVM.

KNN:

- The KNN models demonstrated lower performance than expected with the use of different K-values. The model with the K-value of 1 produced a fairly low accuracy and only increased slightly when increased to a K-value of 5. Any further increase of K-value seemed to slightly decrease the accuracy of the model. It can be concluded that the KNN models produced cannot reliably produce accurate predictions of horse health outcomes given that the accuracy scores fail to reach above even 70%.

Assuming that preprocessing was done correctly for both models, it can be concluded that SVM provides a clearer insight than KNN into the classification of horse health outcomes and can therefore offer more reliable predictions that reflect the given data. But this could simply mean that our chosen dataset is not suitable for KNN. SVM tends to identify outliers better and generally performs better with high-dimensional data/more features so this could also be another reason for our SVM models' superior performance.

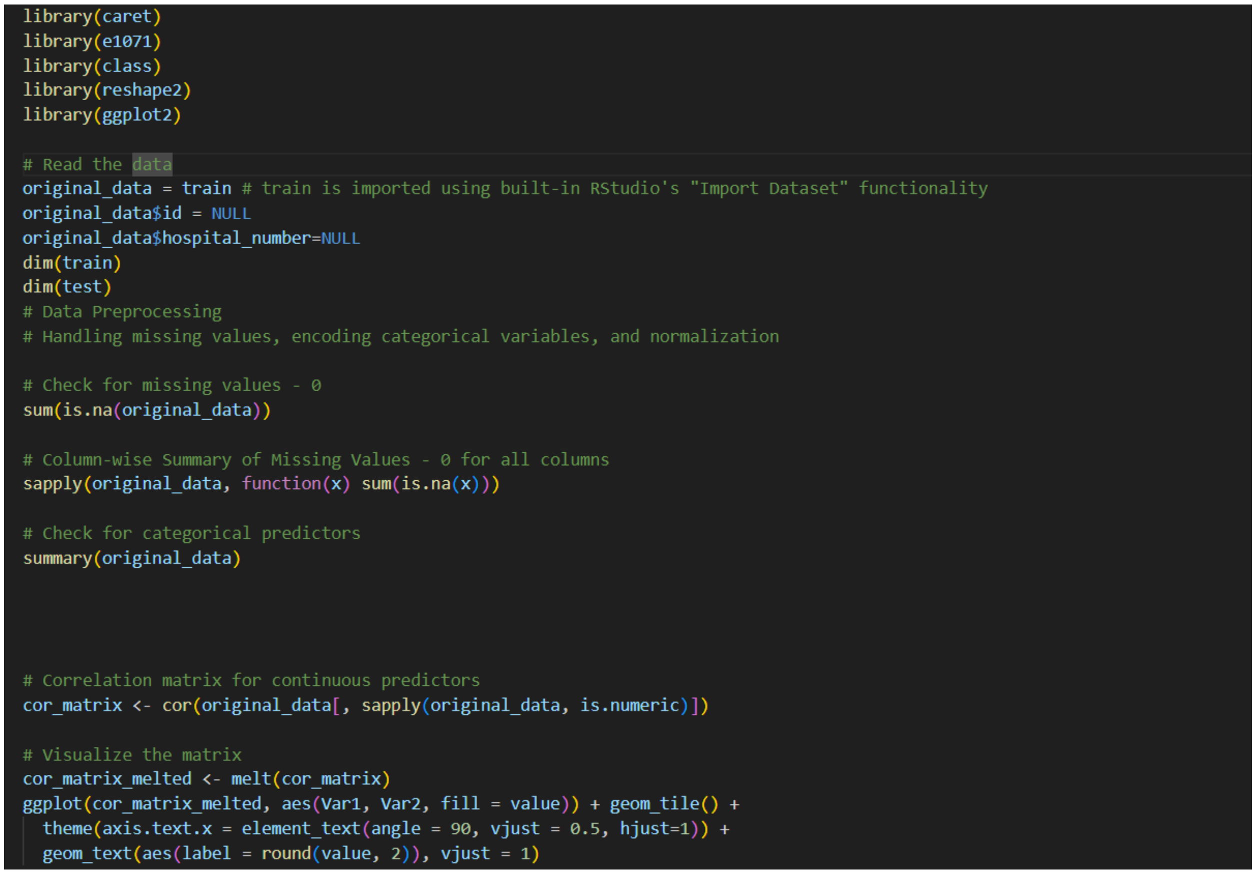
Challenges and Learnings: [Discuss any challenges faced and what you learned.]

- **Data Preparation and variable alignment**: A major inconvenience we ran into from selecting a diverse dataset was there were both quantitative and qualitative variables, which would make it different as we needed extensive preparation to utilize the R functions effectively. An example is changing our categorical variables to numerical in order for it to be used.
- **Hardware Limitations and Model Optimization**: One of the primary challenges in our analysis involved the constraints imposed by limited hardware resources. Given the computationally intensive nature of Support Vector Machines (SVM), especially during hyperparameter tuning, we adopted several strategies to ensure efficient model optimization within these constraints.
    - **Focused Hyperparameter Range**: Instead of an extensive grid search over a wide range of parameters, we narrowed down our hyperparameter search to a more focused range. This was based on preliminary analyses and literature reviews, which suggested potential ranges of parameters that are more likely to yield optimal results.
    - **Incremental Parameter Adjustment**: We employed a strategy of incremental adjustments, starting with broader steps in parameter values. Once a promising range was identified, we fine-tuned within this narrower range. This approach balanced the breadth and depth of the search with the available computational resources.
    - **Efficient Kernel Selection**: Given the different computational demands of various SVM kernels, we prioritized the exploration of the linear kernel, which is typically less resource-intensive than its polynomial or RBF counterparts. This decision was based on both our dataset's characteristics and the need to manage computational load.

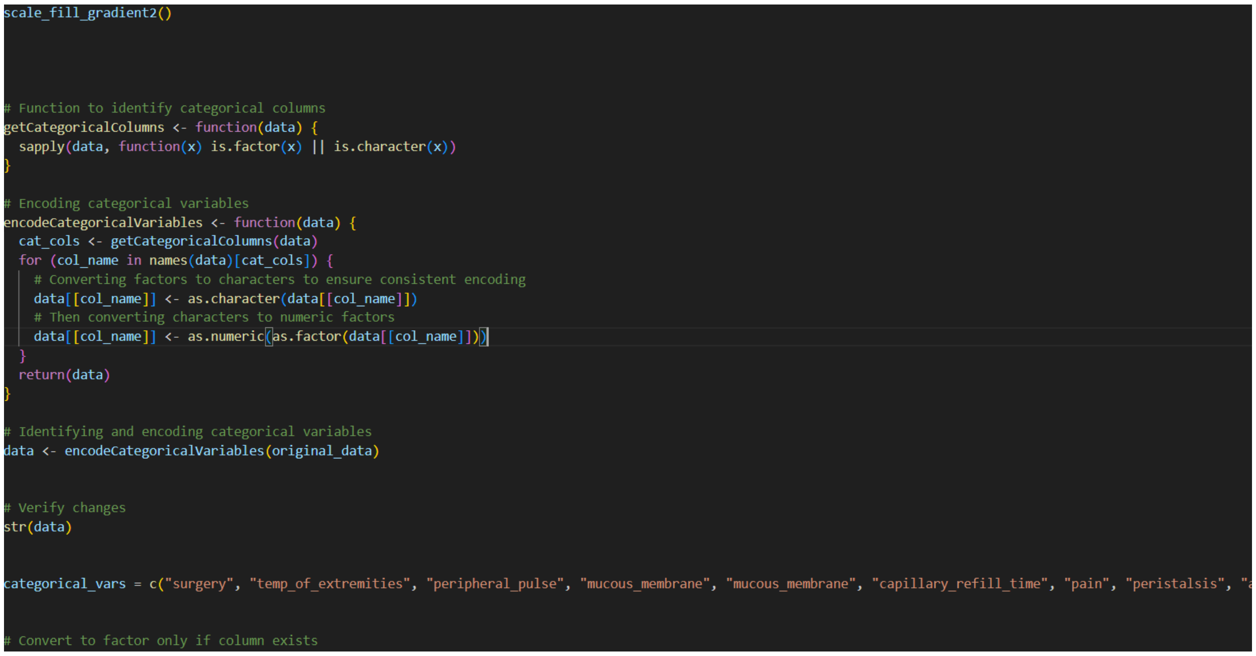
Appendix (Optional)

[Include additional materials like full R code, technical derivations, or extended data tables.]

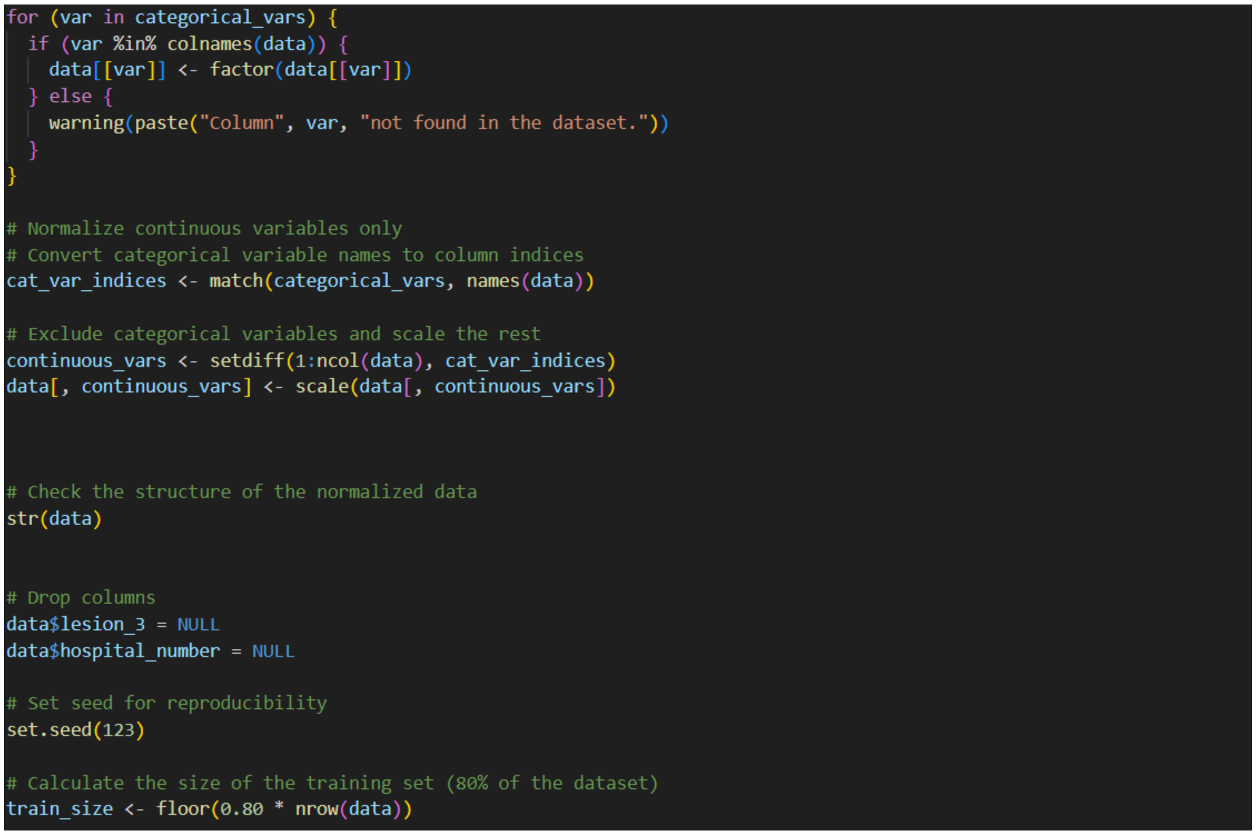
KNN R Code:



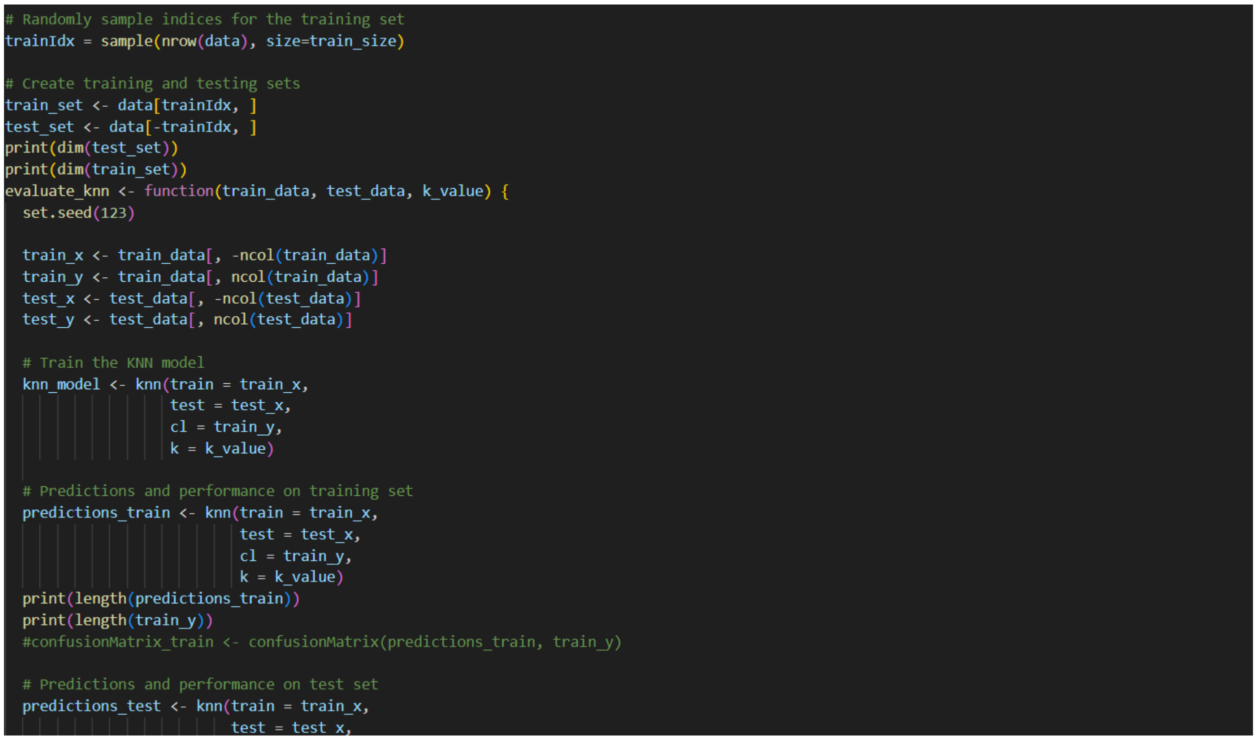
```r
library(caret)
library(e1071)
library(class)
library(reshape2)
library(ggplot2)

# Read the data
original_data = train # train is imported using built-in RStudio's "Import Dataset" functionality
original_data$id = NULL
original_data$hospital_number=NULL
dim(train)
dim(test)
# Data Preprocessing
# Handling missing values, encoding categorical variables, and normalization

# Check for missing values - 0
sum(is.na(original_data))

# Column-wise Summary of Missing Values - 0 for all columns
sapply(original_data, function(x) sum(is.na(x)))

# Check for categorical predictors
summary(original_data)



# Correlation matrix for continuous predictors
cor_matrix <- cor(original_data[, sapply(original_data, is.numeric)])

# Visualize the matrix
cor_matrix_melted <- melt(cor_matrix)
ggplot(cor_matrix_melted, aes(Var1, Var2, fill = value)) + geom_tile() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  geom_text(aes(label = round(value, 2)), vjust = 1)
```



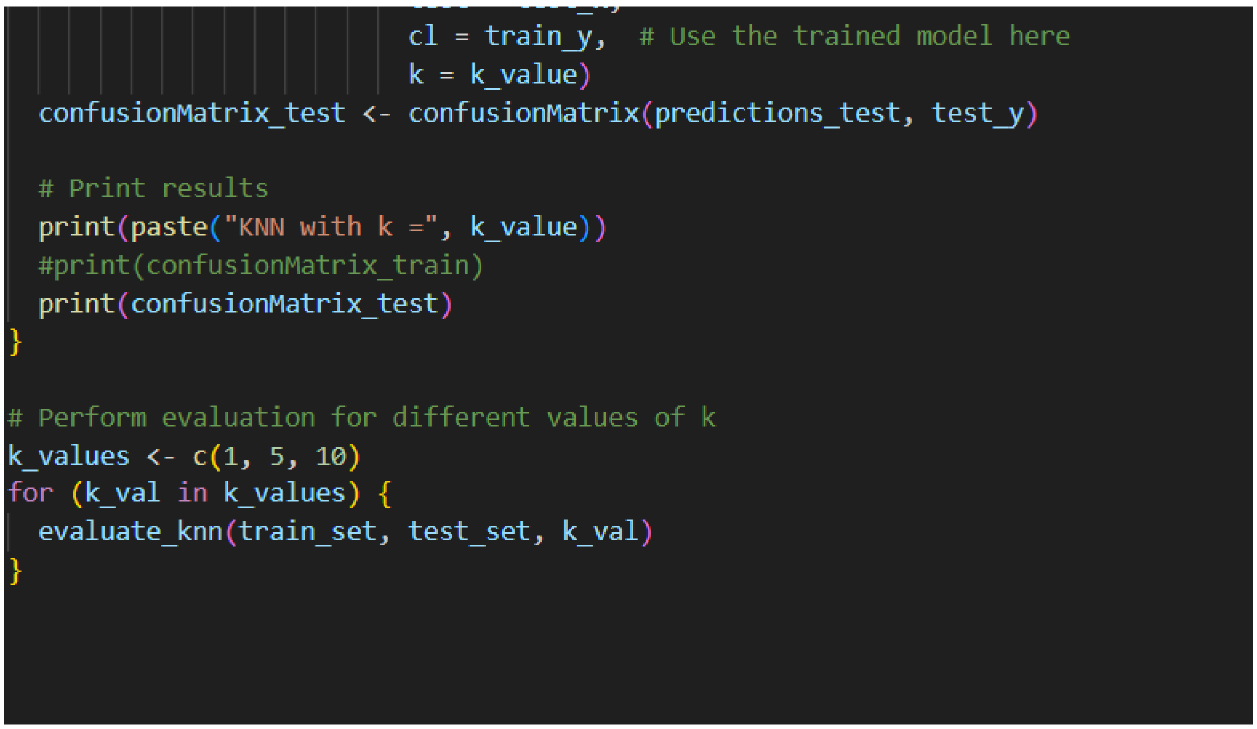
```r
scale_fill_gradient2()



# Function to identify categorical columns
getCategoricalColumns <- function(data) {
  sapply(data, function(x) is.factor(x) || is.character(x))
}

# Encoding categorical variables
encodeCategoricalVariables <- function(data) {
  cat_cols <- getCategoricalColumns(data)
  for (col_name in names(data)[cat_cols]) {
    # Converting factors to characters to ensure consistent encoding
    data[[col_name]] <- as.character(data[[col_name]])
    # Then converting characters to numeric factors
    data[[col_name]] <- as.numeric(as.factor(data[[col_name]]))
  }
  return(data)
}

# Identifying and encoding categorical variables
data <- encodeCategoricalVariables(original_data)


# Verify changes
str(data)


categorical_vars = c("surgery", "temp_of_extremities", "peripheral_pulse", "mucous_membrane", "mucous_membrane", "capillary_refill_time", "pain", "peristalsis", "a


# Convert to factor only if column exists
```



```r
for (var in categorical_vars) {
  if (var %in% colnames(data)) {
    data[[var]] <- factor(data[[var]])
  } else {
    warning(paste("Column", var, "not found in the dataset."))
  }
}

# Normalize continuous variables only
# Convert categorical variable names to column indices
cat_var_indices <- match(categorical_vars, names(data))

# Exclude categorical variables and scale the rest
continuous_vars <- setdiff(1:ncol(data), cat_var_indices)
data[, continuous_vars] <- scale(data[, continuous_vars])



# Check the structure of the normalized data
str(data)


# Drop columns
data$lesion_3 = NULL
data$hospital_number = NULL

# Set seed for reproducibility
set.seed(123)

# Calculate the size of the training set (80% of the dataset)
train_size <- floor(0.80 * nrow(data))
```



```r
# Randomly sample indices for the training set
trainIdx = sample(nrow(data), size=train_size)

# Create training and testing sets
train_set <- data[trainIdx, ]
test_set <- data[-trainIdx, ]
print(dim(test_set))
print(dim(train_set))
evaluate_knn <- function(train_data, test_data, k_value) {
  set.seed(123)

  train_x <- train_data[, -ncol(train_data)]
  train_y <- train_data[, ncol(train_data)]
  test_x <- test_data[, -ncol(test_data)]
  test_y <- test_data[, ncol(test_data)]

  # Train the KNN model
  knn_model <- knn(train = train_x,
                   test = test_x,
                   cl = train_y,
                   k = k_value)

  # Predictions and performance on training set
  predictions_train <- knn(train = train_x,
                           test = test_x,
                           cl = train_y,
                           k = k_value)
  print(length(predictions_train))
  print(length(train_y))
  #confusionMatrix_train <- confusionMatrix(predictions_train, train_y)

  # Predictions and performance on test set
  predictions_test <- knn(train = train_x,
                          test = test_x,
```



```r
                        cl = train_y,  # Use the trained model here
                        k = k_value)
  confusionMatrix_test <- confusionMatrix(predictions_test, test_y)

  # Print results
  print(paste("KNN with k =", k_value))
  #print(confusionMatrix_train)
  print(confusionMatrix_test)
}

# Perform evaluation for different values of k
k_values <- c(1, 5, 10)
for (k_val in k_values) {
  evaluate_knn(train_set, test_set, k_val)
}
```

SVM R Code (Cong):

```r
install.packages("e1071")
install.packages("caret")
library(caret)
library(e1071)
library(reshape2)

# Read the data
original_data = train # train is imported using built-in RStudio's "Import Dataset" functionality
original_data$id = NULL
original_data$hospital_number=NULL


# Data Preprocessing
# Handling missing values, encoding categorical variables, and normalization

# Check for missing values - 0
sum(is.na(original_data))

# Column-wise Summary of Missing Values - 0 for all columns
sapply(original_data, function(x) sum(is.na(x)))

# Check for categorical predictors
summary(original_data)




# Correlation matrix for continuous predictors
cor_matrix <- cor(original_data[, sapply(original_data, is.numeric)])

# Visualize the matrix
cor_matrix_melted <- melt(cor_matrix)
ggplot(cor_matrix_melted, aes(Var1, Var2, fill = value)) + geom_tile() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  geom_text(aes(label = round(value, 2)), vjust = 1)
scale_fill_gradient2()


# Function to identify categorical columns
getCategoricalColumns <- function(data) {
  sapply(data, function(x) is.factor(x) || is.character(x))
}

# Encoding categorical variables
encodeCategoricalVariables <- function(data) {
  cat_cols <- getCategoricalColumns(data)
  for (col_name in names(data)[cat_cols]) {
        # Converting factors to characters to ensure consistent encoding
        data[[col_name]] <- as.character(data[[col_name]])
        # Then converting characters to numeric factors
        data[[col_name]] <- as.numeric(as.factor(data[[col_name]]))
  }
  return(data)
}

# Identifying and encoding categorical variables
data <- encodeCategoricalVariables(original_data)


# Verify changes
str(data)


categorical_vars = c("surgery", "temp_of_extremities", "peripheral_pulse", "mucous_membrane",
"mucous_membrane", "capillary_refill_time", "pain", "peristalsis", "abdominal_distention",
"nasogastric_tube", "rectal_exam_feces", "abdomen", "abdomo_appearance", "surgical_lesion", "cp_data",
"lesion_3", "lesion_2", "outcome")


# Convert to factor only if column exists
for (var in categorical_vars) {
  if (var %in% colnames(data)) {
        data[[var]] <- factor(data[[var]])
  } else {
        warning(paste("Column", var, "not found in the dataset."))
  }
}

# Normalize continuous variables only
# Convert categorical variable names to column indices
cat_var_indices <- match(categorical_vars, names(data))

# Exclude categorical variables and scale the rest
continuous_vars <- setdiff(1:ncol(data), cat_var_indices)
data[, continuous_vars] <- scale(data[, continuous_vars])

# Check the structure of the normalized data
str(data)


# Drop columns
data$lesion_3 = NULL
data$hospital_number = NULL

# Set seed for reproducibility
set.seed(123)

# Calculate the size of the training set (80% of the dataset)
train_size <- floor(0.80 * nrow(data))

# Randomly sample indices for the training set
sample(nrow(data), size=train_size)
```

```r
# Create training and testing sets
train_set <- data[trainIdx, ]
test_set <- data[-trainIdx, ]


# SVM Model Training with hyperparameter tuning

# Define a function to perform SVM tuning and evaluation
evaluate_svm <- function(kernel_type, train_data, test_data, cost_values, gamma_values=NULL,
degree_values=NULL) {
  set.seed(123)
  control <- tune.control(cross = 10)
  # Tune the model
  if(!is.null(gamma_values)){
        tune_result <- tune(svm, outcome ~ ., data = train_data, kernel = kernel_type, ranges =
list(cost = cost_values, gamma = gamma_values), tunecontrol = control)
  }
  else if(!is.null(degree_values)){
        tune_result <- tune(svm, outcome ~ ., data = train_data, kernel = kernel_type, ranges =
list(cost = cost_values, degree = degree_values), tunecontrol = control)
  }
  else {
        tune_result <- tune(svm, outcome ~ ., data = train_data, kernel = kernel_type, ranges =
list(cost = cost_values), tunecontrol = control)
  }


  # Train the model with the best parameters
  best_model <- tune_result$best.model

  # Predictions and performance on training set
  predictions_train <- predict(best_model, newdata = train_data)
  confusionMatrix_train <- confusionMatrix(predictions_train, train_data$outcome)

  # Predictions and performance on test set
  predictions_test <- predict(best_model, newdata = test_data)
  confusionMatrix_test <- confusionMatrix(predictions_test, test_data$outcome)

  # Print results
  print(kernel_type)
  print(summary(tune_result))
  print(confusionMatrix_train)
  print(confusionMatrix_test)
}

# Perform evaluation for each kernel type
evaluate_svm("linear", train_set, test_set, cost_values = c(0.1,1,10,100))
# evaluate_svm("radial", train_set, test_set, cost_values = c(), gamma_values = seq(from=0.1, to=0.3,
by=0.01))
evaluate_svm("radial", train_set, test_set, cost_values = c(0.01,0.1,1,10,100,1000), gamma_values =
c(0.01,0.1,1,10))
evaluate_svm("polynomial", train_set, test_set, cost_values = seq(1, 10, 1), degree_values = c(1, 2))
```