the hyperplane equation is $0.5 - 3x_1 + x_2 = 0$.

if they are support vector, the points should be on the margin boundary line.

for x_1 ($\frac{1}{6}$, $\frac{3}{2}$, 1). $0.5 - 3 \times \frac{1}{6} + \frac{3}{2} = \frac{3}{2} > 1$.

x_1 is not on the hyperplane, either not on margin boundary,

so it is not support vector.

for x_2 ($\frac{1}{3}$, $-\frac{1}{2}$, -1). $0.5 - 3 \times \frac{1}{3} - \frac{1}{2} = -1$

x_2 is support vector.

for x_3 ($\frac{1}{2}$, 2, 1) $0.5 - 3 \times \frac{1}{2} + 2 = 1$

x_3 is support vector

for x_4 ($\frac{1}{3}$, 1, 1). $0.5 - 3 \times \frac{1}{3} + 1 = 0.5 < 1$

x_4 is not on hyperplane, either not on margin boundary. so it is not support vector.

for x_5 ($\frac{1}{4}$, $\frac{3}{4}$, -1) $0.5 - 3 \times \frac{1}{4} + \frac{3}{4} = 0.5 < 1$.

x_5 is not on hyperplane, either not on margin boundary. so it is not support vector.

so, only x_2, x_3 are the support vector of hyperplane $\{x_1, x_2 : 0.5 - 3x_1 + x_2 = 0\}$

1). using euclidean distance to calculate every points to two

centroid $u_1^{(t)}$ and $u_2^{(t)}$.

$u_1^{(t)} = (0.62, 0.06, 1.63)$ $\qquad u_2^{(t)} = (-0.11, 0.92, 0.02)$

for points $X_i = (x_{i1}, x_{i2}, x_{i3})$, we can calculate distance:

$$d(X_i, u_1^{(t)}) = \sqrt{(x_{i1} - 0.62)^2 + (x_{i2} - 0.06)^2 + (x_{i3} - 1.63)^2}$$

$$d(X_i, u_2^{(t)}) = \sqrt{(x_{i1} + 0.11)^2 + (x_{i2} - 0.92)^2 + (x_{i3} - 0.02)^2}$$

so. for $X = \begin{bmatrix} 1.76 & -1.30 & 2.39 \\ 2.41 & 0.66 & 0.41 \\ -2.63 & 1.18 & -0.36 \\ 0.82 & 0.64 & 1.66 \\ -0.73 & 0.83 & 0.85 \end{bmatrix}$

so, $d(x_1, u_1^{(t)}) = \sqrt{(1.76 - 0.62)^2 + (-1.30 - 0.06)^2 + (2.39 - 1.63)^2} = 1.93049$

$d(x_1, u_2^{(t)}) = \sqrt{(1.76 + 0.11)^2 + (-1.30 - 0.92)^2 + (2.39 - 0.02)^2} = 3.74729$

$d(x_2, u_1^{(t)}) = \sqrt{(2.41 - 0.62)^2 + (0.66 - 0.06)^2 + (0.4 - 1.63)^2} = 2.24778$

$d(x_2, u_2^{(t)}) = \sqrt{(2.41 + 0.11)^2 + (0.66 - 0.92)^2 + (0.4 - 0.02)^2} = 2.56322$

$d(x_3, u_1^{(t)}) = \sqrt{(-2.63 - 0.62)^2 + (1.18 - 0.06)^2 + (-0.36 - 1.63)^2} = 3.97203$

$d(x_3, u_2^{(t)}) = \sqrt{(-2.63+0.11)^2 + (1.18-0.92)^2 + (-0.36-0.02)^2} = 2.56172$

$d(x_4, u_1^{(t)}) = \sqrt{(0.82-0.62)^2 + (0.64-0.06)^2 + (1.66-1.63)^2} = 0.61425$

$d(x_4, u_2^{(t)}) = \sqrt{(0.82+0.11)^2 + (0.64-0.92)^2 + (1.66-0.02)^2} = 1.80889$

$d(x_5, u_1^{(t)}) = \sqrt{(-0.73-0.62)^2 + (0.83-0.06)^2 + (0.85-1.63)^2} = 1.73891$

$d(x_5, u_2^{(t)}) = \sqrt{(-0.73+0.11)^2 + (0.83-0.92)^2 + (0.85-0.02)^2} = 1.03990.$

$d(x_1, u_1^{(t)}) < d(x_1, u_2^{(t)})$, so we give x_1 to cluster 1. $d(x_2, u_1^{(t)}) < d(x_2, u_2^{(t)})$, so we give x_2 to cluster 1.

$d(x_3, u_1^{(t)}) > d(x_3, u_2^{(t)})$, so we give x_3 to cluster 2. $d(x_4, u_1^{(t)}) < d(x_4, u_2^{(t)})$, so we give x_4 to cluster 1.

$d(x_5, u_1^{(t)}) > d(x_5, u_2^{(t)})$, so we give x_5 to cluster 2.

and now we know for cluster 1, we have 3 points.

its $X_1 = \begin{bmatrix} 1.76 & -1.30 & 2.39 \\ 2.41 & 0.66 & 0.41 \\ 0.82 & 0.64 & 1.66 \end{bmatrix}$

So the new centroid $u_1^{(t+1)} = \left(\dfrac{\sum x_{i1}}{n_1}, \dfrac{\sum x_{i2}}{n_1}, \dfrac{\sum x_{i3}}{n_1} \right) = (1.663, 0, 1.487)$

and for cluster 2, we have 2 points.

it's $X_2 = \begin{bmatrix} -2.63 & 1.18 & -0.36 \\ -0.73 & 0.83 & 0.85 \end{bmatrix}$

So the new centroid $u_2^{(t+1)} = \left(\dfrac{\sum x_{i1}}{n_2}, \dfrac{\sum x_{i2}}{n_2}, \dfrac{\sum x_{i3}}{n_3} \right) = (-1.68, 1.005, 0.245)$

2. for the cluster sum of squares moving from iteration t to iteration $t+1$:

for iteration t:

points in cluster 1: x_1, x_4, x_5 $u_1^{(t)} = (0.62, 0.06, 1.63)$
points in cluster 2: x_2, x_3, $u_2^{(t)} = (-0.11, 0.92, 0.02).$

So for cluster 1:

$WCSS_1 = (x_{11} - u_{11}^{(t)})^2 + (x_{12} - u_{12}^{(t)})^2 + (x_{13} - u_{13}^{(t)})^2 + (x_{51} - u_{11}^{(t)})^2 + (x_{52} - u_{12}^{(t)})^2 + (x_{53} - u_{13}^{(t)})^2 + (x_{41} - u_{11}^{(t)})^2 + (x_{42} - u_{12}^{(t)})^2$
$+ (x_{43} - u_{13}^{(t)})^2$

$= (1.76 - 0.62)^2 + (-1.3 - 0.06)^2 + (2.39 - 1.63)^2 + (-0.73 - 0.62)^2 + (0.83 - 0.06)^2 + (0.85 - 1.63)^2 + (0.82 - 0.62)^2 + (0.64 - 0.06)^2$

$= 7.1279$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad + (1.66 - 1.63)^2$

for cluster 2:

$WCSS_2 = (x_{21} - u_{21}^{(t)})^2 + (x_{22} - u_{22}^{(t)})^2 + (x_{23} - u_{23}^{(t)})^2 + (x_{31} - u_{21}^{(t)})^2 + (x_{32} - u_{22}^{(t)})^2 + (x_{33} - u_{23}^{(t)})^2$

$= (2.44 + 0.11)^2 + (0.66 - 0.92)^2 + (0.41 - 0.02)^2 + (-2.63 + 0.11)^2 + (1.18 - 0.92)^2 + (-0.36 - 0.02)^2$

$= 13.1325$

for iteration $t+1$:

points in cluster 1: X_1, X_2, X_4 $u_1^{(t+1)} = (1.663, 0, 1.48)$.

points in cluster 2: X_3, X_5. $u_2^{(t+1)} = (-1.68, 1.005, 0.245)$

for cluster 1:

$WCSS_1 = (x_{11} - u_{11}^{(t+1)})^2 + (x_{12} - u_{12}^{(t+1)})^2 + (x_{13} - u_{13}^{(t+1)})^2 + (x_{21} - u_{11}^{(t+1)})^2 + (x_{22} - u_{12}^{(t+1)})^2 + (x_{23} - u_{13}^{(t+1)})^2 + (x_{41} - u_{11}^{(t+1)})^2$
$\qquad + (x_{42} - u_{12}^{(t+1)})^2 + (x_{43} - u_{13}^{(t+1)})^2$

$\qquad = (1.76 - 1.663)^2 + (-1.3 - 0)^2 + (2.39 - 1.48)^2 + (2.41 - 1.663)^2 + (0.66 - 0)^2 + (0.4 - 1.48)^2 + (0.82 - 1.663)^2$
$\qquad + (0.64 - 0)^2 + (1.66 - 1.48)^2$

$\qquad = 5.818534$

for cluster 2:

$WCSS_2 = (x_{31} - u_{21}^{(t+1)})^2 + (x_{32} - u_{22}^{(t+1)})^2 + (x_{33} - u_{23}^{(t+1)})^2 + (x_{51} - u_{21}^{(t+1)})^2 + (x_{52} - u_{22}^{(t+1)})^2 + (x_{53} - u_{23}^{(t+1)})^2$
$\qquad = (-2.13 + 1.68)^2 + (1.18 - 1.005)^2 + (-0.36 - 0.245)^2 + (-0.73 + 1.68)^2 + (0.83 - 1.005)^2 + (0.85 - 0.245)^2$
$\qquad = 2.5983.$

WCSS of cluster 1 decreased from 7.1279 to 5.818534, which indicates that the points in cluster 1 are more tightly clustered around the new centroid.

WCSS of cluster 2 decreased significantly from 13.1325 to 2.5983, which indicates points in cluster 2 are clustered more closely after updating the centroid.

Assignment2_SML

1. Implement at least three different supervised learning methods to predict the winner of the election in a county based on the numerical input features. Utilize an appropriate framework to compare and tune the different methods considered, evaluating and discussing their relative merits. Select the best model for predicting the winner from the available numerical input features.[50 marks]

```r
load("data_assignment_2.Rdata")
data <- elections
# summary(data)

sum(is.na(data))
```

```
[1] 0
```

```r
data_new <- data[, -c(1, 2)] # Remove 'state' and 'county' columns
data_new$winner <- factor(data_new$winner, levels = c("trump","biden")) # Organize factor data

# Standardize numerical columns
num_cols <- sapply(data_new, is.numeric)  # Determine which columns are numeric
data_new[, num_cols] <- scale(data_new[, num_cols])  # Standardize only the numeric columns
```

In the data preprocessing stage, I first examined the characteristics of the data, identifying what each column represented and observed them accordingly. Then, I checked for missing values and found none, after which I removed the categorical variables "state" and "county," since geographic information is generally not used as input in numerical prediction models. Therefore, I deleted these two columns and selected other numerical data as my dataset. Next, I recoded the "winner" column into a factor variable and explicitly specified the factor levels, with "Trump" as the first level, serving as the reference category. Subsequently, I standardized the remaining columns to prepare for fitting machine learning models later on. I will use logistic regression, SVM, random forest, and kNN algorithms. Standardization helps accelerate the convergence speed of the algorithms.

```r
# separate training data and test data
set.seed(23206703)
train_indices <- sample(1:nrow(data_new), 0.8 * nrow(data_new))
train_data <- data_new[train_indices, ]
test_data <- data_new[-train_indices, ]
```

```r
library(caret)
```

```
Loading required package: ggplot2
```

```
Loading required package: lattice
```

```r
library(pROC)  # for AUC calculation
```

```
Type 'citation("pROC")' for a citation.


Attaching package: 'pROC'

The following objects are masked from 'package:stats':

    cov, smooth, var
```

```r
# Predict and evaluate each model's performance on the test set
train_predictors <- train_data[,-ncol(train_data)]
train_response <- train_data$winner

# Initialize a list to store models
models <- list()
performance <- data.frame(Model = character(), Accuracy = numeric(), AUC = numeric())

# Set trainControl object
control <- trainControl(
  method = "cv",
  number = 10,
  savePredictions = "final",
  classProbs = TRUE,
  summaryFunction = twoClassSummary
)
```

```r
# model tuning and model fitting
# Logistic Regression
set.seed(23206703)
glmGrid <- expand.grid(alpha = c(0, 0.5, 1),
  lambda = 10^seq(-4, -1, length = 10))
suppressWarnings({
  models$Logistic <- train(train_predictors, train_response, method = 'glm',
                      family = 'binomial',
                      trControl = trainControl(method = "cv", number = 10))
})



# Support Vector Machine (SVM)
set.seed(23206703)
svmGrid <- expand.grid(.sigma = seq(0.01, 0.1, length = 5), .C = seq(1, 100, length = 5))
models$SVM <- train(train_predictors, train_response, method='svmRadial',
                trControl=control, tuneGrid=svmGrid, preProcess = c("center", "scale"))
```

```
Warning in train.default(train_predictors, train_response, method =
"svmRadial", : The metric "Accuracy" was not in the result set. ROC will be
used instead.
```

```r
# Random Forest
set.seed(23206703)
rfGrid <- expand.grid(.mtry = c(2, sqrt(ncol(train_predictors)), ncol(train_predictors)/3))
```

```
control <- trainControl(method="cv", number=10)
models$RandomForest <- train(train_predictors, train_response, method='rf',
                             trControl=control, tuneGrid=rfGrid)




# K-Nearest Neighbors (KNN)
set.seed(23206703)
knnGrid <- expand.grid(.k = seq(1, 21, by = 2))
models$KNN <- train(train_predictors, train_response, method='knn',
                    trControl=control, tuneGrid=knnGrid)
```

```
for (model_name in names(models)) {
    model <- models[[model_name]]

    best_params <- model$bestTune

    print(paste("Best parameters for", model_name, ":", toString(best_params)))
}
```

```
[1] "Best parameters for Logistic : none"
[1] "Best parameters for SVM : 0.01, 50.5"
[1] "Best parameters for RandomForest : 9"
[1] "Best parameters for KNN : 11"
```

After parameter tuning for different supervised learning models, I obtained the best parameters for each model. The logistic regression model did not use additional regularization parameters for adjustment, indicating that the default settings might already be sufficient. The optimal parameters for the support vector machine were sigma=0.01 and C=50.5, suggesting that the model needs to balance between fitting the data tightly and maintaining generalization ability. For the random forest, an mtry of 9 indicates that selecting fewer features when building decision trees can achieve better performance. For the K-nearest neighbors, a k of 11 shows that this number of neighbors provides the best classification accuracy. Through these tuning results, each model has been optimized to perform optimally on specific datasets.

```
# Predict and evaluate each model's performance on the test set
test_predictors <- test_data[,-ncol(test_data)]
test_response <- test_data$winner

# Add ROC library for AUC calculation
library(pROC)

# Initialize a data frame to store all performance metrics
performance <- data.frame(Model = character(),
                          Accuracy = numeric(),
                          AUC = numeric(),
                          F1score = numeric(),
                          Sensitivity = numeric(),
                          Specificity = numeric())

# Loop through each model, make predictions, and calculate performance metrics
for(model_name in names(models)) {
```

```r
  # Predict using the current model
  predictions <- predict(models[[model_name]], newdata = test_predictors)

  # Calculate confusion matrix
  confusion_matrix <- confusionMatrix(predictions, test_response)
  accuracy <- confusion_matrix$overall['Accuracy']
  sensitivity <- confusion_matrix$byClass['Sensitivity']
  specificity <- confusion_matrix$byClass['Specificity']
  f1_score <- confusion_matrix$byClass['F1']

  # Calculate AUC
  roc_result <- roc(response = test_response, predictor = as.numeric(predictions),
                    levels = rev(levels(test_response)))
  auc <- auc(roc_result)

  # Append to the performance data frame
  performance <- rbind(performance, data.frame(Model = model_name,
                                               Accuracy = accuracy,
                                               AUC = auc,
                                               F1score = f1_score,
                                               Sensitivity = sensitivity,
                                               Specificity = specificity))

 }
```

```
Setting direction: controls > cases
Setting direction: controls > cases
Setting direction: controls > cases
Setting direction: controls > cases
```

```r
 # Print the performance metrics
 print(performance)
```

```
              Model  Accuracy       AUC   F1score Sensitivity Specificity
Accuracy    Logistic 0.9867110 0.9747864 0.9921260   0.9921260   0.9574468
Accuracy1        SVM 0.9767442 0.9428715 0.9863014   0.9921260   0.8936170
Accuracy2 RandomForest 0.9883721 0.9757706 0.9931170   0.9940945   0.9574468
Accuracy3        KNN 0.9651163 0.9056375 0.9795918   0.9921260   0.8191489
```

```r
 # Optional: Select and print the best model based on Test Accuracy or AUC
 best_model <- performance[which.max(performance$Accuracy),]
 cat("The best model on test set is", best_model$Model,
     "with an Accuracy of", best_model$Accuracy,
     "and AUC of", best_model$AUC,
     "and F1 Score of", best_model$F1score,
     ", Sensitivity of", best_model$Sensitivity,
     ", and Specificity of", best_model$Specificity, "\n")
```

The best model on test set is RandomForest with an Accuracy of 0.9883721 and AUC of 0.9757706 and F1 Score of 0.993117 , Sensitivity of 0.9940945 , and Specificity of 0.9574468

Based on the result data and model performance metrics, the random forest model demonstrated the best performance on the test set, specifically showing a very high accuracy (0.9883721), excellent AUC value (0.9757706), and an extremely high F1 score (0.993117).

Additionally, the model's sensitivity (0.9940945) and specificity (0.9574468) also reached high levels, indicating good balance. These results suggest that the random forest model exhibits high accuracy and reliability in distinguishing between "Trump" and "Biden" as winners.

Compared to other models, such as logistic regression, support vector machine (SVM), and K-nearest neighbors (KNN), the random forest holds a distinct advantage in most key performance indicators. For instance, the logistic regression model had an accuracy of 0.9867110 and AUC of 0.9747864, which, although close to that of the random forest, was slightly lower in the F1 score. The performance of the SVM was somewhat weaker, with an accuracy of 0.9767442, AUC of 0.9428715, and a specificity of only 0.8936170, indicating a lesser ability to discriminate between different candidates' victories. The KNN model performed the lowest among these models, with an accuracy of 0.9651163, AUC of 0.9056375, and a specificity of just 0.8191489, which could be due to KNN's sensitivity to high-dimensional data and noise.

In summary, the random forest model not only surpasses other models in overall accuracy and AUC performance but also demonstrates higher performance in balancing error types (i.e., sensitivity and specificity). This makes the random forest the best choice for predicting election outcomes in distinguishing between "Trump" and "Biden" as winners.

```r
library(ggplot2)
library(tidyr)  # For pivot_longer

performance_long <- pivot_longer(performance,
                                 cols = c(Accuracy, F1score, Sensitivity, Specificity),
                                 names_to = "Metric",
                                 values_to = "Value")

performance_long$Model <- factor(performance_long$Model, levels = unique(performance_long$Mode

ggplot(performance_long, aes(x = Model, y = Value, fill = Metric)) +
  geom_bar(stat = "identity", position = position_dodge()) +
  theme_minimal() +
  labs(title = "Comparison of Model Metrics", x = "Model", y = "Metric Value") +
  facet_wrap(~ Metric, scales = "free_y") +
  geom_text(aes(label = sprintf("%.2f", Value)), position = position_dodge(0.9),
            vjust = -0.5, size = 3) +
  coord_flip()
```

Comparison of Model Metrics
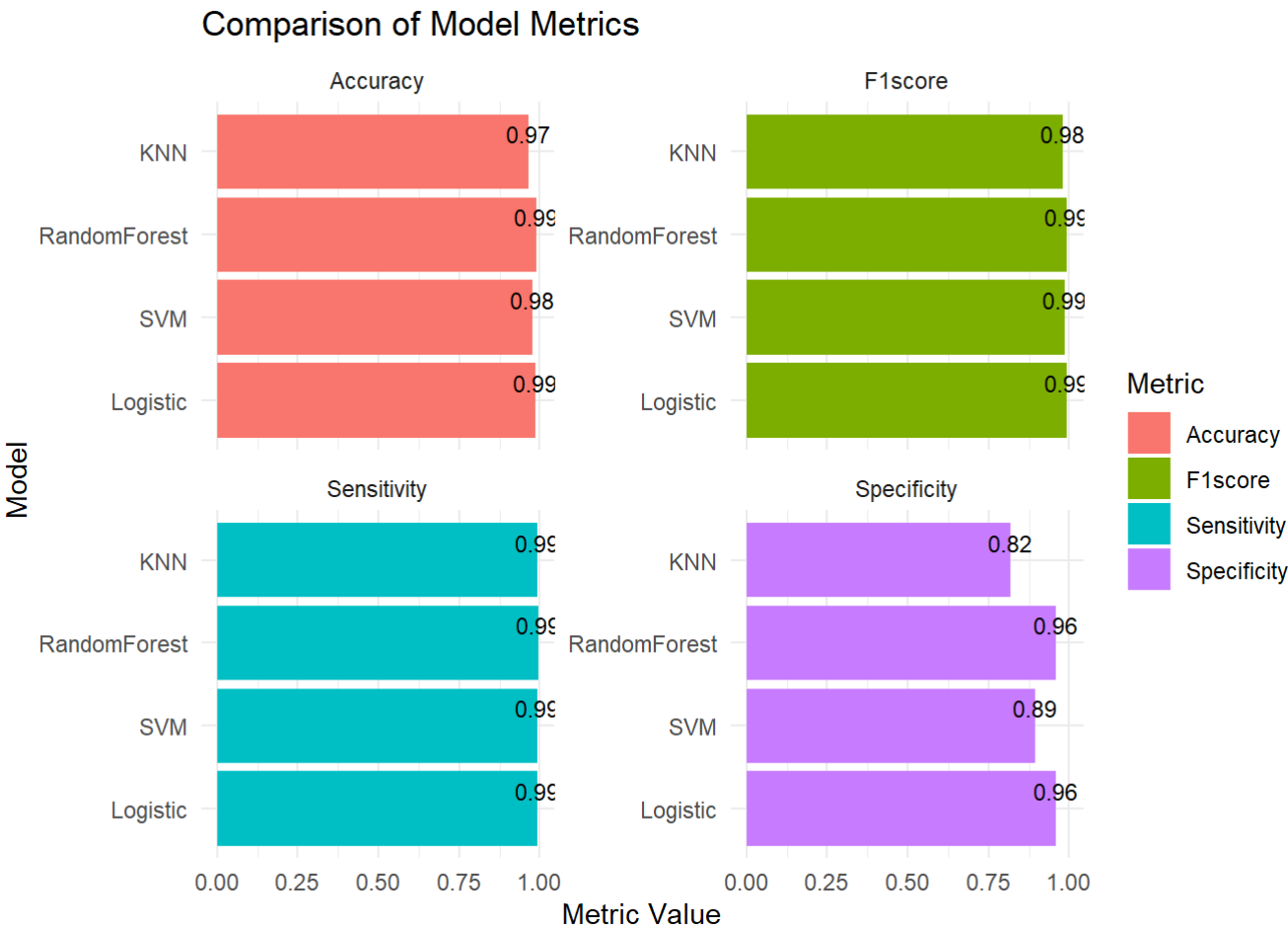
In this visualization, I compare the performance of four different supervised learning models—logistic regression, support vector machine (SVM), random forest, and K-nearest neighbors (KNN)—across various performance metrics.

Accuracy: The random forest and logistic regression models show the highest accuracy, nearly reaching 0.98, indicating that they can predict election outcomes very accurately overall. The accuracy of SVM is slightly lower, while KNN has the lowest accuracy.

F1 Score: In terms of the F1 score, the random forest and logistic regression also perform excellently, scoring over 0.99, which indicates that these models achieve a good balance between precision and recall. The F1 score of SVM follows closely, whereas KNN is significantly lower.

Sensitivity: All models display high sensitivity, especially the random forest and logistic regression, which are nearly perfect. This suggests that these models are very effective at identifying the positive class (such as correctly predicting the election winner as "Trump" or "Biden").

Specificity: In terms of specificity, logistic regression and random forest also show higher values, meaning they can effectively identify the negative class (i.e., correctly predicting non-winners). KNN performs the worst on this metric, which may indicate challenges in distinguishing the negative class.

Overall, the random forest and logistic regression models demonstrate very strong performance across all considered performance metrics, almost consistently outperforming SVM and KNN. This suggests that ensemble methods (like random forest) and logistic

regression may be more effective than other algorithms when dealing with election data. The performance of the random forest is particularly notable; it not only does well in correctly classifying the election winners (high sensitivity) but also excels in avoiding misclassification of non-winners as winners (high specificity). These characteristics make the random forest a powerful and reliable option, suitable for predicting election outcomes.

2. Use appropriately some test data in order to evaluate the generalized predictive performance of the best selected classifier. Provide a discussion about the ability of the selected model at predicting if the winner will be biden or trump.[20marks]

```r
# Choose the best model (based on AUC or accuracy)
best_model_name <- performance[which.max(performance$AUC), "Model"]

# Extract the best model
best_model <- models[[best_model_name]]

# Make predictions on the test data using the best model
predictions <- predict(best_model, newdata = test_predictors)
predicted_probabilities <- predict(best_model, newdata = test_predictors, type = "prob")

# Calculate performance metrics
conf_matrix <- confusionMatrix(predictions, test_response)
accuracy <- conf_matrix$overall['Accuracy']
auc_value <- roc(response = test_response, predictor = predicted_probabilities[, "trump"])$auc
```

```
Setting levels: control = trump, case = biden
```

```
Setting direction: controls > cases
```

```r
f1_score <- conf_matrix$byClass['F1']

# Output performance metrics
cat(sprintf("The best model (%s) has an Accuracy of %.2f, AUC of %.2f, and F1 Score of %.2f.\n
            best_model_name, accuracy, auc_value, f1_score))
```

```
The best model (RandomForest) has an Accuracy of 0.99, AUC of 1.00, and F1 Score of 0.99.
```

```r
# Visualize the predicted probability distribution
ggplot(data.frame(Actual = test_response, Predicted_Probability =
                  predicted_probabilities[, "trump"]),
       aes(x = Actual, y = Predicted_Probability)) +
  geom_boxplot(aes(color = Actual)) +
  labs(title = sprintf("Predicted Probability Distribution for Trump (Model: %s)",
                       best_model_name),
       x = "Actual Category",
       y = "Predicted Probability for Trump") +
  theme_minimal()
```

Predicted Probability Distribution for Trump (Model: RandomForest)
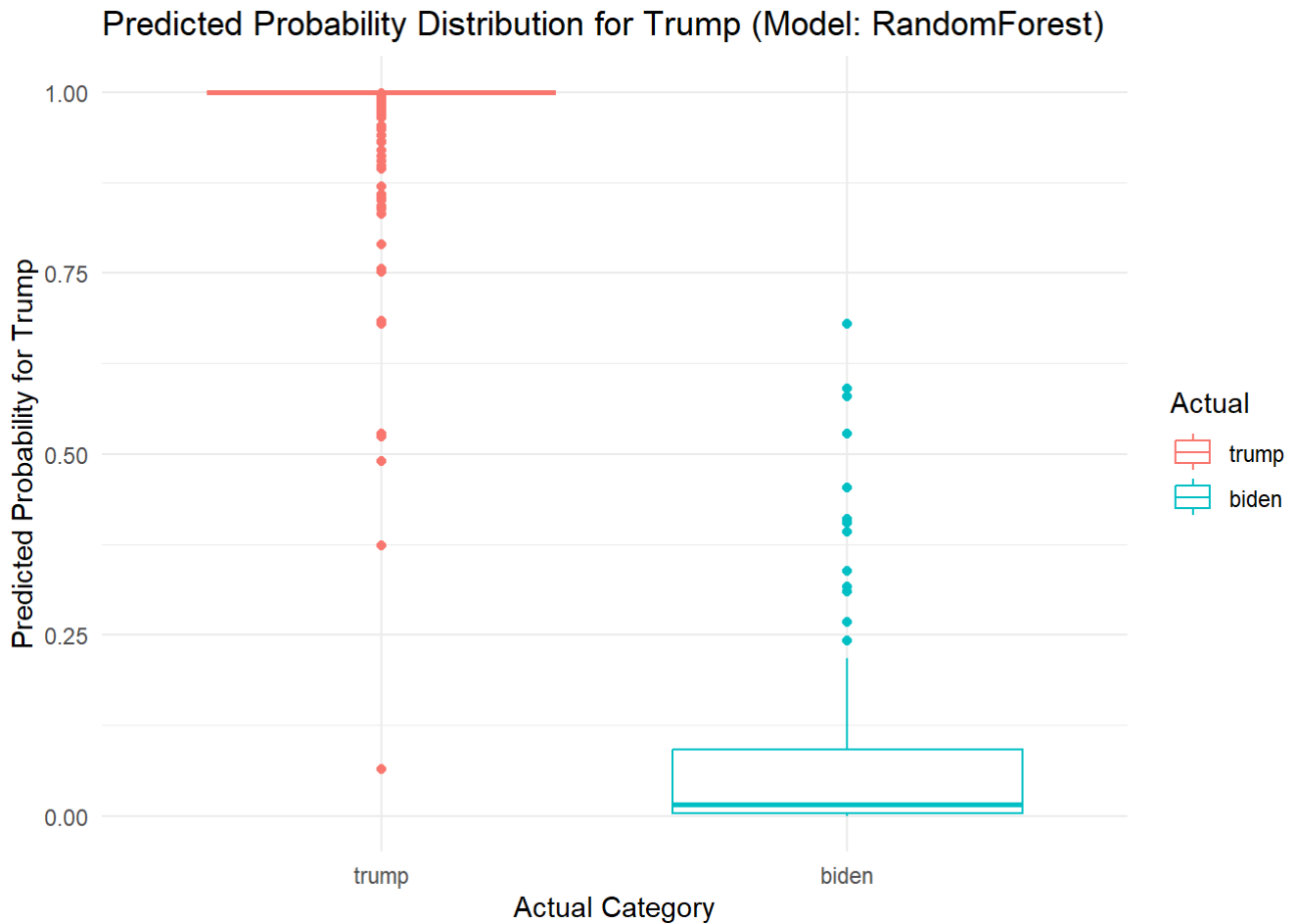
From the content above, we can see that the random forest model fits the data best, having the highest AUC and accuracy. The model's performance metrics show an accuracy (Accuracy) of 0.99, an AUC value (Area Under Curve) of 1.00, and an F1 score of 0.99. These metrics are close to perfect, indicating that the model has very high predictive performance on the test set.

From the box plot, we can see the distribution of the probabilities that the random forest model predicts for Trump's victory. For data points where the actual category is Trump, the predicted probabilities are concentrated at high values, almost all close to 1.00, which means that the model is very confident about Trump's chances of winning. For data points where the actual category is Biden, the majority of predicted probabilities are distributed in the lower area, close to 0.00, indicating that the model can effectively recognize situations where Biden is less likely to win.