final

Group 9

2024-05-03

```
calculate_accuracy_with_tolerance <- function(predictions, actual, tolerance) {
  # Ensure both inputs are numeric
  if(is.factor(predictions)) {
    predictions <- as.numeric(as.character(predictions))
  }
  if(is.factor(actual)) {
    actual <- as.numeric(as.character(actual))
  }

  # Calculate accuracy with tolerance
  correct_predictions <- abs(predictions - actual) <= tolerance
  accuracy <- sum(correct_predictions) / length(actual) * 100
  return(accuracy)
}
```

```
set.seed(2)

# Reading dataset
# Load data
wine_quality <- read.csv("/Users/a./Downloads/wine-quality-white-and-red.csv")

# Convert quality to a factor
wine_quality$quality <- as.factor(wine_quality$quality)

# Data preprocessing
library(caret)

# One-hot encoding for the 'type' column
dummies <- dummyVars(~ type, data = wine_quality)
wine_quality_transformed <- predict(dummies, newdata = wine_quality)
wine_quality <- cbind(wine_quality[, !(names(wine_quality) %in% "type")], wine_quality_transformed)

# Scale data excluding the quality and one-hot encoded columns
preProcValues <- preProcess(wine_quality[, !names(wine_quality) %in% c("quality", "type.red", "type.whi
wine_quality[, !names(wine_quality) %in% c("quality", "type.red", "type.white")] <- predict(preProcValu

# Creating interaction terms
wine_quality$interaction1 <- with(wine_quality, total.sulfur.dioxide * free.sulfur.dioxide)
wine_quality$interaction2 <- with(wine_quality, alcohol * volatile.acidity)

# Polynomial features
wine_quality$fixed.acidity2 <- wine_quality$fixed.acidity^2
```

```r
wine_quality$citric.acid2 <- wine_quality$citric.acid^2

# Splitting data into training, validation, and test set
set.seed(123) # Set seed for reproducibility
n <- nrow(wine_quality)
train_index <- sample(1:n, 0.7 * n)
remaining_index <- setdiff(1:n, train_index)
val_index <- sample(remaining_index, 0.15 * n)
test_index <- setdiff(remaining_index, val_index)

train <- wine_quality[train_index, ]
validation <- wine_quality[val_index, ]
test <- wine_quality[test_index, ]

# Function to calculate accuracy
calculate_accuracy <- function(predictions, actual) {
  correct_predictions <- predictions == actual
  accuracy <- sum(correct_predictions) / length(actual) * 100
  return(accuracy)
}

# Model fitting and validation
# Logistic Regression
library(nnet)
logistic_fit <- multinom(quality ~ ., data = train)
```

```
## # weights:  133 (108 variable)
## initial  value 8848.053448
## iter  10 value 5344.873608
## iter  20 value 5214.748125
## iter  30 value 5110.980972
## iter  40 value 4987.868096
## iter  50 value 4918.036084
## iter  60 value 4829.644618
## iter  70 value 4796.539071
## iter  80 value 4766.285223
## iter  90 value 4744.059117
## iter 100 value 4732.777432
## final  value 4732.777432
## stopped after 100 iterations
```

```r
logistic_pred_validation <- predict(logistic_fit, validation, type = "class")
cat("Logistic Regression Accuracy: ", calculate_accuracy(logistic_pred_validation, validation$quality),
```

```
## Logistic Regression Accuracy:  54.72279
```

```r
cat("Logistic Regression Accuracy with Tolerance 1.0: ", calculate_accuracy_with_tolerance(logistic_pre
```

```
## Logistic Regression Accuracy with Tolerance 1.0:  94.35318 %
```

```r
# Random Forest
library(randomForest)
rf_fit <- randomForest(quality ~ ., data = train, ntree = 100)
rf_pred_validation <- predict(rf_fit, validation)
cat("Random Forest Accuracy: ", calculate_accuracy(rf_pred_validation, validation$quality), "\n")
```

```
## Random Forest Accuracy:  67.24846
```

```r
cat("Random Forest Accuracy with Tolerance 1.0: ", calculate_accuracy_with_tolerance(rf_pred_validation
```

```
## Random Forest Accuracy with Tolerance 1.0:  96.71458 %
```

```r
# Support Vector Machine
library(e1071)
svm_pred_validation <- predict(svm_fit, validation)
cat("SVM Accuracy: ", calculate_accuracy(svm_pred_validation, validation$quality), "\n")
```

```
## SVM Accuracy:  63.03901
```

```r
cat("SVM Accuracy with Tolerance 1.0: ", calculate_accuracy_with_tolerance(svm_pred_validation, validati
```

```
## SVM Accuracy with Tolerance 1.0:  96.09856 %
```

```r
library(caret)
library(nnet)  # Required for multinom

# Prepare the data
train_scaled <- predict(preProcValues, train[, -ncol(train)])
validation_scaled <- predict(preProcValues, validation[, -ncol(train)])

# Add scaled data back to datasets
train_scaled <- cbind(train_scaled, quality = train$quality)
validation_scaled <- cbind(validation_scaled, quality = validation$quality)

# Define training control for cross-validation
train_control <- trainControl(
  method = "cv",
  number = 10,  # Number of cross-validation folds
  savePredictions = "final",
  verboseIter = FALSE
)

# Set up a grid of hyperparameters to tune
# Adjusting decay values to see which works best
decay_grid <- expand.grid(decay = c(0.01, 0.05, 0.1, 0.5))

# Fit logistic regression with hyperparameter tuning
logistic_model <- train(
  quality ~ .,
  data = train_scaled,
```

```
  method = "multinom",  # Using multinomial logistic regression from nnet
  trControl = train_control,
  tuneGrid = decay_grid,
  maxit = 200  # Max iterations for convergence
)
```

```
## # weights:  126 (102 variable)
## initial  value 7962.664330
## iter  10 value 5518.402711
## iter  20 value 5274.358347
## iter  30 value 5173.699039
## iter  40 value 5074.773198
## iter  50 value 5005.608371
## iter  60 value 4746.989970
## iter  70 value 4492.785492
## iter  80 value 4311.423961
## iter  90 value 4286.921197
## iter 100 value 4283.073733
## iter 110 value 4279.004725
## iter 120 value 4276.927310
## iter 130 value 4275.697901
## iter 140 value 4275.487589
## iter 150 value 4275.407732
## iter 160 value 4275.330671
## iter 170 value 4275.245492
## iter 180 value 4275.210111
## iter 190 value 4275.198775
## iter 200 value 4275.189770
## final  value 4275.189770
## stopped after 200 iterations
## # weights:  126 (102 variable)
## initial  value 7962.664330
## iter  10 value 5518.403318
## iter  20 value 5274.367684
## iter  30 value 5173.775651
## iter  40 value 5075.178703
## iter  50 value 5006.310746
## iter  60 value 4748.915593
## iter  70 value 4516.005575
## iter  80 value 4352.463660
## iter  90 value 4312.147762
## iter 100 value 4307.927583
## iter 110 value 4303.499235
## iter 120 value 4301.877727
## iter 130 value 4300.945127
## iter 140 value 4300.717111
## iter 150 value 4300.568512
## iter 160 value 4300.356507
## iter 170 value 4300.266163
## iter 180 value 4300.211792
## iter 190 value 4300.205065
## iter 200 value 4300.201010
## final  value 4300.201010
```

```r
# Predict on validation set with the best model
logistic_pred_validation <- predict(logistic_model, validation_scaled)

# Calculate accuracies
basic_accuracy <- calculate_accuracy_with_tolerance(logistic_pred_validation, validation_scaled$quality
accuracy_tolerance_10 <- calculate_accuracy_with_tolerance(logistic_pred_validation, validation_scaled$

# Display accuracies
cat("Tuned Logistic Regression Accuracy: ", basic_accuracy, "%\n")
```

```
## Tuned Logistic Regression Accuracy:  54.72279 %
```

```r
cat("Tuned Logistic Regression Accuracy with Tolerance 1.0: ", accuracy_tolerance_10, "%\n")
```

```
## Tuned Logistic Regression Accuracy with Tolerance 1.0:  94.66119 %
```

```r
# Store accuracies for comparison
lr_accuracy_tolerance_0 <- basic_accuracy
lr_accuracy_tolerance_1 <- accuracy_tolerance_10
```

```r
# Fine tuning random forest
library(caret)
library(lattice)
library(ggplot2)
fitControl <- trainControl(
  method = "cv",           # Use k-fold cross-validation
  number = 10,             # Number of folds
  savePredictions = "final",
  verboseIter = TRUE,
  allowParallel = TRUE    # Allow parallel processing
)

# Define the tuning grid
rfGrid <- expand.grid(
  mtry = c(2, sqrt(ncol(train) - 1), round(ncol(train)/3))  # Suggested values for 'mtry'
)

# Training the model
rfModel <- train(
  quality ~ .,                # formula
  data = train,               # training data
  method = "rf",              # model type
  trControl = fitControl,     # training control
  tuneGrid = rfGrid,          # the tuning grid
  metric = "Accuracy"         # performance metric
)
```

```
## + Fold01: mtry=2.000
## - Fold01: mtry=2.000
## + Fold01: mtry=4.123
## - Fold01: mtry=4.123
```

```
## + Fold10: mtry=6.000
## - Fold10: mtry=6.000
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 4.12 on full training set
```

```
# Predicting on validation set
predictions <- predict(rfModel, newdata = validation)

# Calculate accuracy
validation_accuracy <- calculate_accuracy_with_tolerance(predictions, validation$quality, 0)

validation_accuracy_tolerance_10 <- calculate_accuracy_with_tolerance(predictions, validation$quality,

# Print accuracies
cat("Validation Accuracy:", validation_accuracy, "%\n")
```

```
## Validation Accuracy: 67.65914 %
```

```
cat("Validation Accuracy with Tolerance 1.0:", validation_accuracy_tolerance_10, "%\n")
```

```
## Validation Accuracy with Tolerance 1.0: 96.61191 %
```

```
rf_accuracy_tolerance_0 <- validation_accuracy
rf_accuracy_tolerance_1 <- validation_accuracy_tolerance_10
```

```
# Fine tuning SVM

library(e1071)

# Define the tuning grid
tuning_grid <- expand.grid(cost = seq(5, 15, by = 2), gamma = c(0.75, 1, 1.25))

# Data frame to store accuracies
accuracies <- data.frame(cost = numeric(), gamma = numeric(), accuracy_0 = numeric(), accuracy_1 = nume

for(i in 1:nrow(tuning_grid)) {
  svm_fit <- svm(quality ~ ., data = train, type = "C-classification", kernel = "radial",
                 cost = tuning_grid$cost[i], gamma = tuning_grid$gamma[i])
  svm_pred_validation <- predict(svm_fit, validation)
  accuracy_0 <- calculate_accuracy_with_tolerance(svm_pred_validation, validation$quality, 0)
  accuracy_1 <- calculate_accuracy_with_tolerance(svm_pred_validation, validation$quality, 1)

  accuracies <- rbind(accuracies, data.frame(cost = tuning_grid$cost[i], gamma = tuning_grid$gamma[i],
                                             accuracy_0 = accuracy_0, accuracy_1 = accuracy_1))
}

best_settings_0 <- accuracies[which.max(accuracies$accuracy_0),]
best_settings_1 <- accuracies[which.max(accuracies$accuracy_1),]
print(paste("Best settings for accuracy_0: Cost=", best_settings_0$cost, "Gamma=", best_settings_0$gamma
```

```
## [1] "Best settings for accuracy_0: Cost= 7 Gamma= 0.75 with Accuracy= 63.7577002053388"
```

```
print(paste("Best settings for accuracy_1: Cost=", best_settings_1$cost, "Gamma=", best_settings_1$gamma
```

```
## [1] "Best settings for accuracy_1: Cost= 15 Gamma= 1 with Accuracy= 96.2012320328542"
```

```
svm_accuracy_tolerance_0 <-best_settings_0$accuracy_0
svm_accuracy_tolerance_1 <- best_settings_1$accuracy_1
```

```
library(xgboost)
train[, -ncol(train)] <- sapply(train[, -ncol(train)], as.numeric)
validation[, -ncol(validation)] <- sapply(validation[, -ncol(validation)], function(x) {
  if(is.factor(x) || is.character(x)) as.numeric(as.character(x)) else x
})
# Ensure labels start from 0
train_labels <- as.numeric(as.character(train$quality)) - 1

validation_labels <- as.numeric(as.factor(validation$quality)) - 1

unique_train_labels <- sort(unique(train_labels))
unique_validation_labels <- sort(unique(validation_labels))
# Checking the range of labels and setting num_class accurately
num_classes <- max(c(unique_train_labels, unique_validation_labels)) + 1  # max label + 1

# Creating the matrix for XGBoost ensuring that only numeric columns are included
train_matrix <- xgb.DMatrix(data = as.matrix(train[, -ncol(train)]), label = train_labels)
validation_matrix <- xgb.DMatrix(data = as.matrix(validation[, -ncol(validation)]), label = validation_

params <- list(
    booster = "gbtree",
    objective = "multi:softmax",
    num_class = num_classes,
    eval_metric = "mlogloss",
    eta = 0.1,
    max_depth = 6,
    subsample = 0.8,
    colsample_bytree = 0.8,
    min_child_weight = 1
)

# Training with XGBoost
xgb_fit <- xgb.train(
    params = params,
    data = train_matrix,
    nrounds = 100,
    watchlist = list(train = train_matrix, eval = validation_matrix),
    early_stopping_rounds = 10
)
```

```
## [1]  train-mlogloss:1.609611 eval-mlogloss:2.002844
## Multiple eval metrics are present. Will use eval_mlogloss for early stopping.
## Will train until eval_mlogloss hasn't improved in 10 rounds.
##
## [2]  train-mlogloss:1.427305 eval-mlogloss:2.004882
```

```
## [3]  train-mlogloss:1.301917 eval-mlogloss:1.989387
## [4]  train-mlogloss:1.175381 eval-mlogloss:1.989532
## [5]  train-mlogloss:1.054759 eval-mlogloss:1.983959
## [6]  train-mlogloss:0.930223 eval-mlogloss:2.045305
## [7]  train-mlogloss:0.827428 eval-mlogloss:2.112423
## [8]  train-mlogloss:0.757563 eval-mlogloss:2.132371
## [9]  train-mlogloss:0.698549 eval-mlogloss:2.129074
## [10] train-mlogloss:0.626493 eval-mlogloss:2.189583
## [11] train-mlogloss:0.560597 eval-mlogloss:2.256717
## [12] train-mlogloss:0.504651 eval-mlogloss:2.325689
## [13] train-mlogloss:0.453080 eval-mlogloss:2.390070
## [14] train-mlogloss:0.416459 eval-mlogloss:2.429050
## [15] train-mlogloss:0.390423 eval-mlogloss:2.404189
## Stopping. Best iteration:
## [5]  train-mlogloss:1.054759 eval-mlogloss:1.983959
```

```
# Prediction
predictions <- predict(xgb_fit, validation_matrix)

# Evaluate Accuracy
xgb_accuracy_tolerance_0 <- calculate_accuracy_with_tolerance(predictions, validation_labels, 0)
xgb_accuracy_tolerance_1 <- calculate_accuracy_with_tolerance(predictions, validation_labels, 1)

cat("Accuracy with Tolerance 0: ", xgb_accuracy_tolerance_0, "%\n")
```

```
## Accuracy with Tolerance 0:  2.566735 %
```

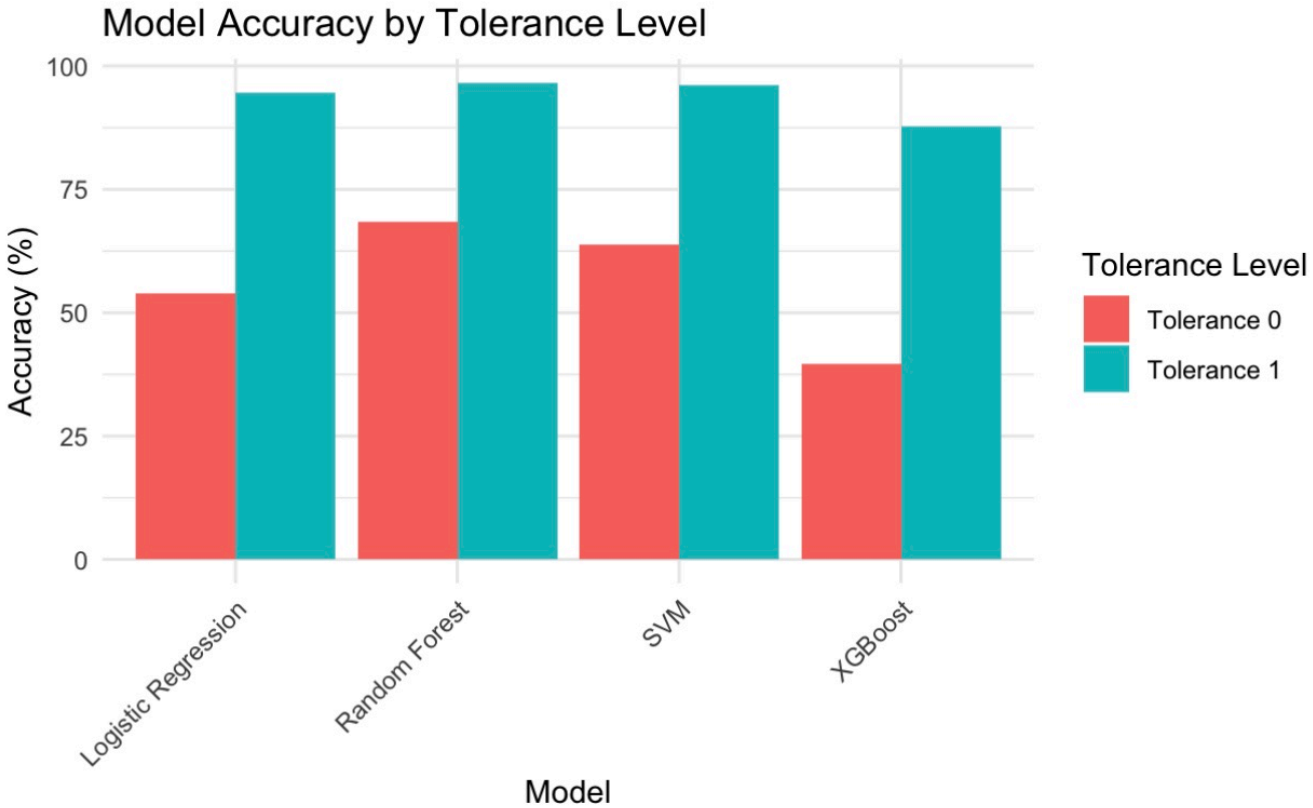
```
cat("Accuracy with Tolerance 1: ", xgb_accuracy_tolerance_1, "%\n")
```

```
## Accuracy with Tolerance 1:  18.89117 %
```

```
model_data <- data.frame(
  Model = rep(c("Random Forest", "SVM", "Logistic Regression", "XGBoost"), each = 2),
  Tolerance = rep(c("Tolerance 0", "Tolerance 1"), times = 4),
  Accuracy = c(
    rf_accuracy_tolerance_0, rf_accuracy_tolerance_1,
    svm_accuracy_tolerance_0, svm_accuracy_tolerance_1,
    lr_accuracy_tolerance_0, lr_accuracy_tolerance_1,
    xgb_accuracy_tolerance_0, xgb_accuracy_tolerance_1
  )
)
```

```
library(ggplot2)

# Plotting the accuracies
ggplot(model_data, aes(x = Model, y = Accuracy, fill = Tolerance)) +
  geom_bar(stat = "identity", position = position_dodge()) +
  labs(title = "Model Accuracy by Tolerance Level",
       x = "Model",
       y = "Accuracy (%)",
       fill = "Tolerance Level") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```


Model Accuracy by Tolerance Level
100
75
50
25
0
Accuracy (%)
Logistic Regression
Random Forest
SVM
XGBoost
Model
Tolerance Level
Tolerance 0
Tolerance 1