Project Report
Intro to Statistical Computing
STAT 610
Chess Outcome Prediction

Introduction

The goal of the project is to create a predictive model that predicts the outcome of the chess game based on the information before the game such as the players' ratings, the type of match, the tournament. The data set is available at https://www.kaggle.com/datasnaek/chess. The data set contains information about 20,000 chess games.

Objectives

- To create a predictive model that predicts the outcome of the chess game based on the information before the game such as the players' ratings, the type of match, the tournament.
- To create a pipeline that uses the data and performs the following steps:
  - Data cleaning
  - Data exploration
  - Data visualization
  - Model building
  - Model evaluation

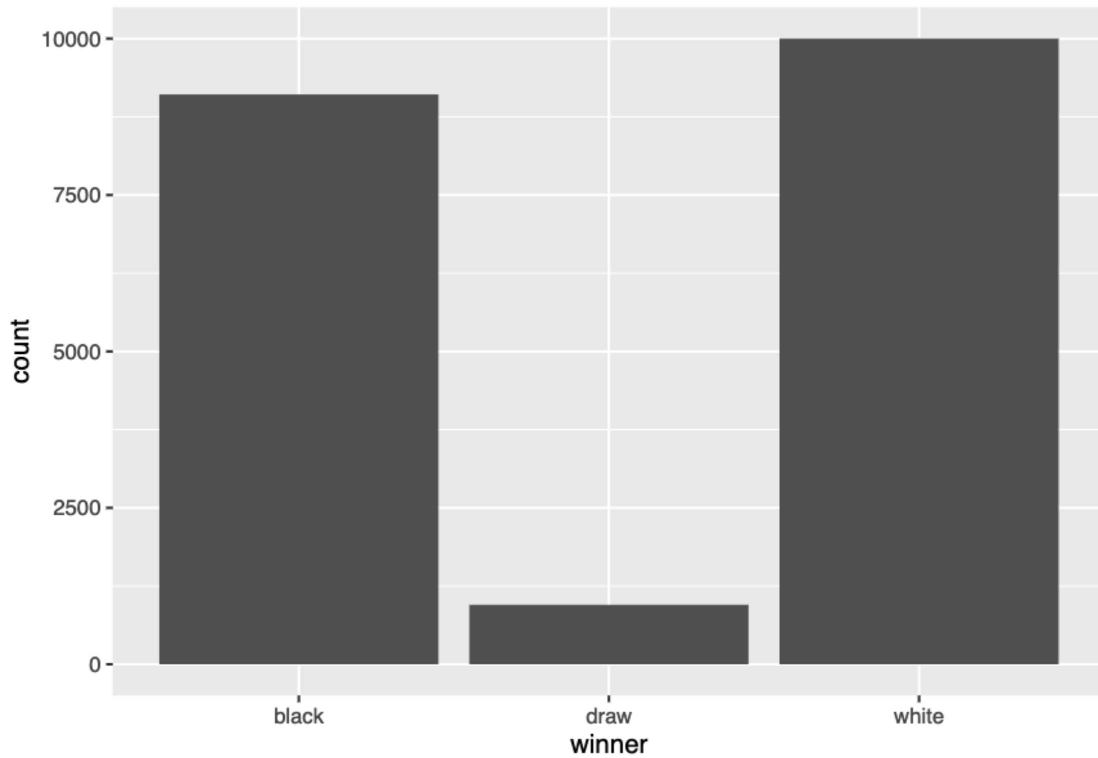

Fig 1: Distribution of the outcome of the matches in the dataset.

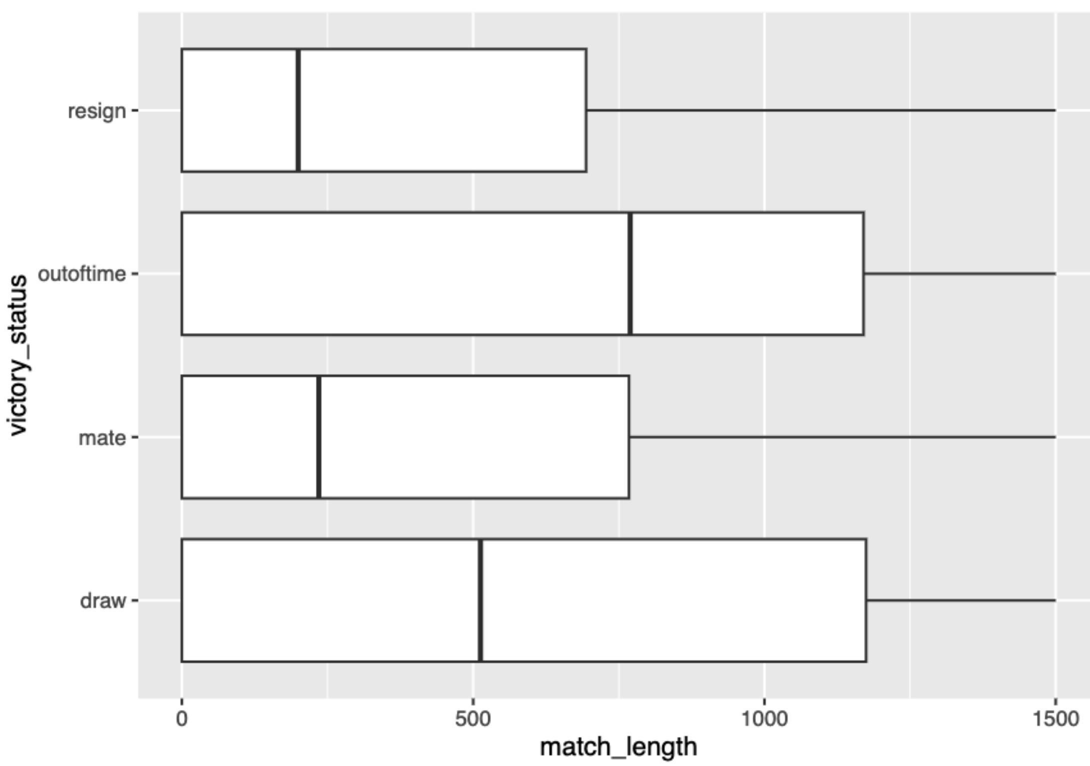

Fig 2: Distribution of match length (in seconds) vs the outcome of the match

Design Decisions

I have created a EDA notebook that performs the following steps:

1. Data Wrangling
   a. Data importing
   b. Checking the data types of variables and performing necessary steps to change the data type based on relevancy.
   c. Univariate Statistics of the variables
   d. Dealing with null values
   e. Outlier analysis
2. Data Visualization
   a. Bivariate analysis
   b. Correlation analysis
3. Feature
   a. Chi square test
   b. ANOVA test
   c. Correlation test
4. Linear Regression
   a. Linear Regression assumption validation
   b. Model training
   c. Hyperparameter tuning
   d. Regularization
   e. Result visualization

I have created a file "model_training.R" that performs the following steps:
- Prepare the data for model training
- Perform iterative feature selection for categorical and numerical variables
- Train the model using the selected features

```r
preprocess <- function(data) {
    chess <- data
    chess$rated <- as.logical(chess$rated)
    chess$increment_code <- as.character(chess$increment_code)
    # everthing before the plus sign
    chess$increment_code1 <- strsplit(chess$increment_code, split = "\\+")
    chess$increment_code1 <- sapply(chess$increment_code1, function(x) x[1])
    # everything after the plus sign
    chess$increment_code2 <- strsplit(chess$increment_code, split = "\\+")
    chess$increment_code2 <- sapply(chess$increment_code2, function(x) x[2])

    chess$increment_code1 <- as.numeric(chess$increment_code1)
    chess$increment_code2 <- as.numeric(chess$increment_code2)

    chess$increment_code1_bin <- cut(chess$increment_code1, breaks = c(0,
    20, 50, 100, 200, 1000))
    chess$increment_code2_bin <- cut(chess$increment_code2, breaks = c(0, 20,
    50, 100, 200, 1000))
    data <- chess
    data$rating_diff <- data$white_rating - data$black_rating

    # Remove the draws
    data <- subset(data, winner != "draw")
    # Convert the winner column to 1 and 0
    data$winner <- ifelse(data$winner == "white", 1, 0)
    # Remove the columns that we don't need
    data <- data[, c("winner", "white_rating", "black_rating",
     "rating_diff", "increment_code1_bin", "increment_code2_bin",
     "rated")]
    # Remove the rows with NA values
    data <- na.omit(data)
    # Return the preprocessed data
    return(data)
}
train_model <- function(data, verbose = FALSE) {
  # Split the data into train and test
  train_index <- createDataPartition(data$winner, p = 0.7, list = FALSE)
  train <- data[train_index, ]
  test <- data[-train_index, ]
  # Train the model
  model <- glm(winner ~ ., data = train, family = "binomial")
  # Predict the test data
  pred <- predict(model, test)
  if (verbose) {
    print(model)
    print(summary(model))
    # Print the confusion matrix
    print(table(test$winner, pred > 0.5))
    # Print the accuracy
    print(mean(test$winner == (pred > 0.5)))

  }
  # return the model and the mean accuracy
  return(list(model = model, accuracy = mean(test$winner == (pred > 0.5))))
}
```

```r
iterative_sampling_num <- function(data, numerical_vars) {
  selected_vars <- c()
  accuracy <- -1
  # Run the below code iteratively while adding variable if
  # the mean accuracy is greater than the previous one
  while (length(numerical_vars) > 0) {
    # Run the chi-square test on the numerical variables and
    # select the variable with the least p-value
    anova_pvals <- sapply(numerical_vars,
      function(x) anova_test(data, "winner", x))
    # Select the variable with the least p-value
    anova_var <- numerical_vars[which.min(anova_pvals)]
    # Run the model with the selected variable with the
    # train_model function which will return the model and the accuracy
    selected_vars <- c(selected_vars, anova_var)
    model <- train_model(data[, c("winner", selected_vars)], verbose = TRUE)
    # If the accuracy is greater than the previous one,
    # then add the variable to the model
    numerical_vars <- numerical_vars[-which(numerical_vars == anova_var)]
    if (model$accuracy > accuracy) {
      accuracy <- model$accuracy
    } else {
      # removez the last variable added to the model
      selected_vars <- selected_vars[-length(selected_vars)]
    }
  }
  return(selected_vars)
}

#iterative sampling for categorical variables
iterative_sampling_cat <- function(data, categorical_vars) {
  selected_vars <- c()
  accuracy <- -1
  # Run the below code iteratively while adding variable if
  # the mean accuracy is greater than the previous one
  while (length(categorical_vars) > 0) {
    # Run the chi-square test on the categorical variables and
    # select the variable with the least p-value
    chi_pvals <- sapply(categorical_vars,
      function(x) chisq_test(data, "winner", x))
    # Select the variable with the least p-value
    chi_var <- categorical_vars[which.min(chi_pvals)]
    # Run the model with the selected variable with the
    # train_model function which will return the model and the accuracy
    selected_vars <- c(selected_vars, chi_var)
    model <- train_model(data[, c("winner", selected_vars)], verbose = TRUE)
    # If the accuracy is greater than the previous one,
    # then add the variable to the model
    categorical_vars <- categorical_vars[-which(categorical_vars == chi_var)]
    if (model$accuracy > accuracy) {
      accuracy <- model$accuracy
    } else {
```

I've also created a file "tests.R" that performs the following tests on the model training pipeline:
- Test the dataset that has been downloaded and has the required columns and data types
- Test the preprocessing function and the outputs of the function
- The chisquare test and the anova test
- The iterative sampling function for both the categorical and numerical variables
- The linear regression model training function

```r
test_preprocess <- function(file) {
    # Load the dataset
    data <- read.csv(file)
    # Preprocess the dataset
    data_pros <- preprocess(data) # nolint
    # Check if the dataset is not empty
    test_that("Dataset is not empty", {
        expect_true(nrow(data_pros) > 0)
    })
    # Check if the dataset has the right number of columns
    test_that("Dataset has the right number of columns", {
        expect_true(ncol(data_pros) == 7)
    })
    test_that("Dataset has the expected columns", {
        expect_true(sum(colnames(data_pros) == c("winner", "white_rating",
         "black_rating", "rating_diff", "increment_code1_bin",
         "increment_code2_bin", "rated")) == 7)
    })
    # Check if the winner column has only 0 and 1
    test_that("Winner column is binary", {
        expect_true(sum(
            data_pros$winner == 0 |
            data_pros$winner == 1) == nrow(data_pros)
            )
    })
    # Check if the rated column has only TRUE and FALSE
    test_that("Rated column is binary", {
        expect_true(sum(
            data_pros$rated == TRUE |
            data_pros$rated == FALSE) == nrow(data_pros)
            )
    })
    # Check if there are no null values
    test_that("No null values", {
        expect_true(sum(is.na(data_pros)) == 0)
    })

}
```

Conclusion

Finally the output is the model that is saved in the "model.RData" file. The EDA is also saved in the pdf output. The full code is also attached in the zip file in the final submission.