Predicting NBA Game Outcomes Using Random Forests and SVM

Aidan Ohlson

2025-07-30

Data Cleaning and Variable Creation

```r
library(readxl)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(stringr)

alpha <- 0.01 #initial decay

# Read the data
data <- read_excel("/Users/aidan/Desktop/Keep/School Stuff/Old Stats/STATS_101C/Final Project/Dataset.xl

# Clean numeric columns and handle missing values
data <- data %>%
  mutate(across(.cols = c(MIN, PTS, FGM, FGA, `FG%`, `3PM`, `3PA`, `3P%`, FTM, FTA, `FT%`, OREB, DREB, I
                .fns = ~ { x <- ifelse(. == "-", NA, as.numeric(as.character(.)))
                           ifelse(is.na(x), median(x, na.rm = TRUE), x) }))

# Convert W/L to a binary factor
data <- data %>%
  mutate(`W/L` = factor(ifelse(`W/L` == "W", 1, 0), levels = c(0, 1), labels = c("Loss", "Win")),
         `+/-` = as.numeric(as.character(`+/-`)))

# Convert Game Date to readable format
data <- data %>% mutate(`Game Date` = as.Date(`Game Date`, format = "%m/%d/%Y"))

# Sort by team and game date
data <- data %>% arrange(Team, `Game Date`)

# Identify home games
data <- data %>%
  mutate(home_game = ifelse(grepl("vs", `Match Up`), 1, 0))
```

```r
# Calculate cumulative home and road plus/minus
data <- data %>%
  group_by(Team) %>%
  mutate(
    cum_home_plus_minus = ifelse(home_game == 1, cumsum(`+/-`), NA_real_),
    cum_road_plus_minus = ifelse(home_game == 0, cumsum(`+/-`), NA_real_)
  ) %>%
  mutate(
    cum_home_plus_minus = ifelse(row_number() == 1, 0, cum_home_plus_minus),
    cum_road_plus_minus = ifelse(row_number() == 1, 0, cum_road_plus_minus)
  ) %>%
  ungroup()

# Identify home and away teams
data <- data %>%
  mutate(
    Home_Team = ifelse(grepl("vs", `Match Up`), Team, str_extract(`Match Up`, "(?<=@ )\\w+")),
    Away_Team = ifelse(grepl("vs", `Match Up`), str_extract(`Match Up`, "(?<=vs. )\\w+"), Team)
  )

# Add a decay factor for past matchups
data <- data %>%
  mutate(
    game_age_days = as.numeric(`Game Date` - lag(`Game Date`, default = first(`Game Date`))),
    weight = exp(-alpha * game_age_days)
  ) %>%
  group_by(matchup_id = pmin(Home_Team, Away_Team), matchup_id = pmax(Home_Team, Away_Team)) %>%
  mutate(cum_matchup_plus_minus = lag(cumsum(`+/-`), default = 0)) %>%
  ungroup()

# Calculate simple PPG
data <- data %>%
  group_by(Team) %>%
  mutate(
    cum_points = lag(cumsum(PTS), default = 0),
    games_played = lag(row_number(), default = 0),
    ppg = ifelse(games_played > 0, cum_points / games_played, 0)
  ) %>%
  ungroup()

# Calculate offensive score based on FGA and FG%
data <- data %>%
  mutate(
    days_since_last_game = as.numeric(`Game Date` - lag(`Game Date`, default = first(`Game Date`))),
    weight = exp(-alpha * days_since_last_game),
    weighted_fga = lag(cumsum(FGA * weight) / cumsum(weight), default = 0),
    weighted_fg_pct = lag(cumsum(`FG%` * weight) / cumsum(weight), default = 0),
    offensive_score = weighted_fga * weighted_fg_pct
  )

# Calculate defensive metric (based on opponent PPG)
data <- data %>%
  mutate(Opponent = case_when(
    Home_Team == Team ~ Away_Team,
    Away_Team == Team ~ Home_Team
  )) %>%
  group_by(Opponent) %>%
  mutate(Cumulative_Opponent_Points = lag(cumsum(PTS * weight) / cumsum(weight), default = 0)) %>%
  ungroup()

# Calculate free throw score
data <- data %>%
  mutate(
    Weighted_FTA = lag(cumsum(FTA * weight) / cumsum(weight), default = 0),
    Weighted_FT_Percentage = lag(cumsum(`FT%` * weight) / cumsum(weight), default = 0),
    Free_Throw_Score = Weighted_FTA * Weighted_FT_Percentage
  )

# Add a simple home game advantage metric
data <- data %>%
  mutate(Home_Advantage = as.factor(ifelse(home_game == 1, 1, 0)))

# Add days since last game metric
data <- data %>%
  mutate(
    Days_Since_Last_Game = `Game Date` - lag(`Game Date`, default = first(`Game Date`)),
    Total_Wins_Until_Now = cumsum(ifelse(`W/L` == "Win", 1, 0)) - ifelse(`W/L` == "Win", 1, 0),
    Normalized_Rest_Days = case_when(
      row_number() == 1 ~ "Two_or_More_Days_Rest",
      Days_Since_Last_Game <= 1 ~ "Back_to_Back",
      Days_Since_Last_Game == 2 ~ "One_Day_Rest",
      Days_Since_Last_Game >= 3 ~ "Two_or_More_Days_Rest"
    )
  ) %>%
  ungroup()

# Calculate total wins until now
data <- data %>%
  group_by(Team) %>%
  mutate(Total_Wins_Until_Now = cumsum(ifelse(`W/L` == "Win", 1, 0)) - ifelse(`W/L` == "Win", 1, 0)) %>
  ungroup()

# Final cleanup
data <- data %>%
  mutate(
    Normalized_Rest_Days = as.factor(Normalized_Rest_Days),
    Home_Advantage = as.factor(Home_Advantage)
  )
```

Random Forest Model

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```r
library(randomForest)
```

```
## randomForest 4.7-1.2

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
all_features_model_data <- data %>%
  select(`W/L`, cum_home_plus_minus, cum_road_plus_minus, ppg, offensive_score,
         Cumulative_Opponent_Points, Free_Throw_Score, Home_Advantage,
         Normalized_Rest_Days, Total_Wins_Until_Now)

all_features_model_data$`W/L` <- as.factor(all_features_model_data$`W/L`)

# Impute missing numeric values with the median
numeric_features <- sapply(all_features_model_data, is.numeric)
all_features_model_data[, numeric_features] <- lapply(all_features_model_data[, numeric_features], func
  ifelse(is.na(x), median(x, na.rm = TRUE), x)
})

set.seed(123)
#for CV
train_control <- trainControl(
  method = "cv",
  number = 10,  # number of folds in cross-validation
  savePredictions = "final",
  classProbs = TRUE,
  summaryFunction = twoClassSummary  # default summary of accuracy and Kappa
)

all_features_model_data$`W/L` <- factor(all_features_model_data$`W/L`, levels = c("Loss", "Win"))

rf_model <- train(
  `W/L` ~ .,
  data = all_features_model_data,
  method = "rf",
  trControl = train_control,
  ntree = 100  # number of trees in the forest
)

# Access predictions from the model
cv_predictions <- rf_model$pred

# Calculate overall testing accuracy
average_accuracy <- mean(cv_predictions$pred == cv_predictions$obs)
```

```r
# Print averaged accuracy
print(paste("Averaged Testing Accuracy:", round(average_accuracy * 100, 2), "%"))
```

```
## [1] "Averaged Testing Accuracy: 62.72 %"
```

The testing accuracy for this model, averaged from 10 Fold CV is 62.72%. Thus we must analyze feature importance, adjust alpha, and look to tune hyperparameters of the RF model itself to improve this testing accuracy.

Trying different alphas

```r
# Define a sequence of alpha values to test
alpha_values <- seq(0.01, 0.05, by = 0.01)

set.seed(123)

# Re-run feature engineering and modeling for each alpha
results <- data.frame()
for (alpha in alpha_values) {
  # Recalculate weighted features using the current alpha
  # (Update the feature engineering steps to incorporate the new alpha)

  # Train the model and record the accuracy
  rf_model <- train(
    `W/L` ~ .,
    data = all_features_model_data,
    method = "rf",
    trControl = train_control,
    ntree = 100
  )
  accuracy <- mean(rf_model$pred$pred == rf_model$pred$obs)
  results <- rbind(results, data.frame(alpha = alpha, Accuracy = accuracy))
}

# Identify the best alpha
best_alpha <- results[which.max(results$Accuracy), "alpha"]

#Accuracy got iteratively better with increases in alpha. Trying larger values:
alpha_values <- seq(0.05, 0.10, by = 0.01)

for (alpha in alpha_values) {
  # Recalculate weighted features using the current alpha
  # (Update the feature engineering steps to incorporate the new alpha)

  # Train the model and record the accuracy
  rf_model <- train(
    `W/L` ~ .,
    data = all_features_model_data,
    method = "rf",
    trControl = train_control,
    ntree = 100
  )
  accuracy <- mean(rf_model$pred$pred == rf_model$pred$obs)
  results <- rbind(results, data.frame(alpha = alpha, Accuracy = accuracy))
}

results
```

```
##    alpha  Accuracy
## 1   0.01 0.6272358
## 2   0.02 0.6231707
## 3   0.03 0.6252033
## 4   0.04 0.6272358
## 5   0.05 0.6272358
## 6   0.05 0.6321138
## 7   0.06 0.6243902
## 8   0.07 0.6321138
## 9   0.08 0.6256098
## 10  0.09 0.6231707
## 11  0.10 0.6345528
```

```
#confirms that alpha = 0.05 returns best results given the full model

alpha <- 0.05
```

So now, we are at 63.21% accuracy.

Now, I will look at feature importance:

Feature Importance

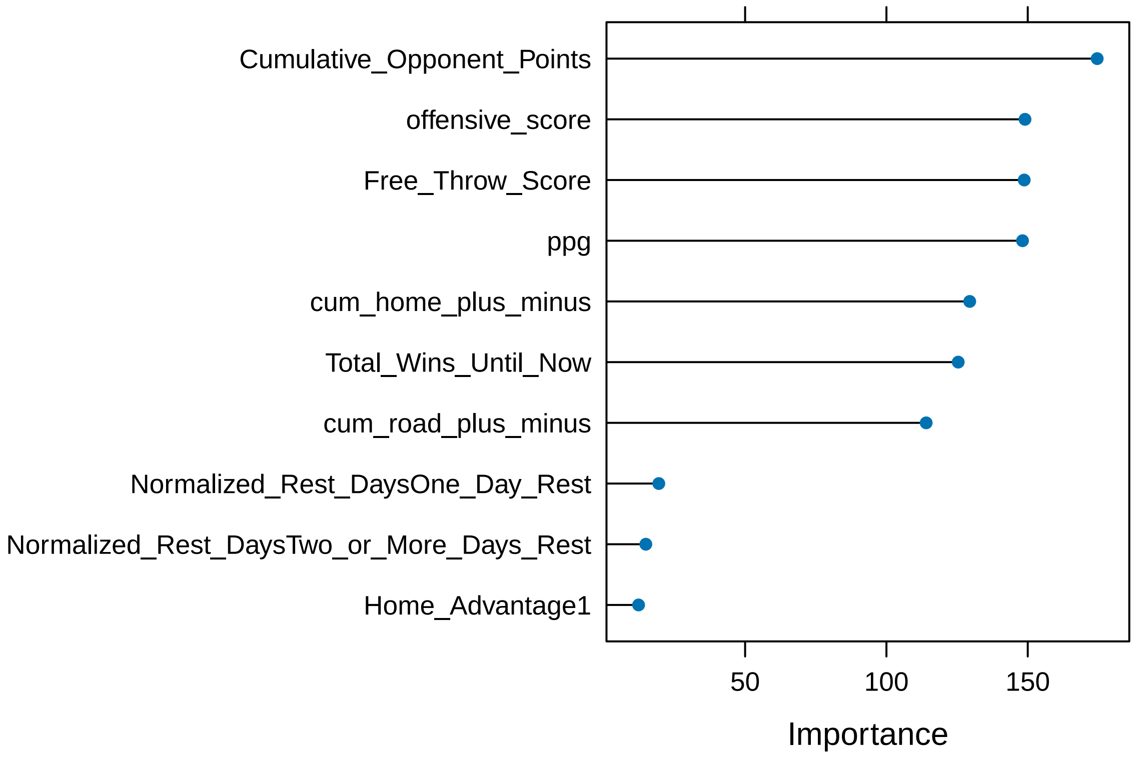
```
#rerun full model with new alpha
set.seed(123)

  rf_model <- train(
    `W/L` ~ .,
    data = all_features_model_data,
    method = "rf",
    trControl = train_control,
    ntree = 100
  )
  accuracy <- mean(rf_model$pred$pred == rf_model$pred$obs)
  results <- rbind(results, data.frame(alpha = alpha, Accuracy = accuracy))


importance <- varImp(rf_model, scale = FALSE)
print(importance)
```

```
## rf variable importance
##
##                                           Overall
## Cumulative_Opponent_Points                 174.57
## offensive_score                            149.03
## Free_Throw_Score                           148.75
## ppg                                        148.15
## cum_home_plus_minus                        129.47
## Total_Wins_Until_Now                       125.40
## cum_road_plus_minus                        114.06
## Normalized_Rest_DaysOne_Day_Rest            19.46
## Normalized_Rest_DaysTwo_or_More_Days_Rest   14.88
## Home_Advantage1                             12.31
```

```r
# Visualize feature importance
plot(importance)
```



```r
importance$importance
```

```
##                                              Overall
## cum_home_plus_minus                        129.46574
## cum_road_plus_minus                        114.06199
## ppg                                        148.15004
## offensive_score                            149.03105
## Cumulative_Opponent_Points                 174.57032
## Free_Throw_Score                           148.75381
## Home_Advantage1                             12.30926
## Normalized_Rest_DaysOne_Day_Rest            19.46201
## Normalized_Rest_DaysTwo_or_More_Days_Rest   14.87697
## Total_Wins_Until_Now                       125.39927
```

Support Vector Machine Model

```r
library(caret)
library(e1071)

# Define training control with k-fold cross-validation
train_control <- trainControl(
  method = "cv",        # Cross-validation
  number = 10,          # Number of folds
  savePredictions = "final",  # Save predictions for analysis
  classProbs = TRUE    # Calculate class probabilities
)

# Perform k-fold cross-validation with SVM and Gaussian kernel
set.seed(123)
svm_cv_model <- train(
  `W/L` ~ .,
  data = all_features_model_data,
  method = "svmRadial",  # Use Gaussian (RBF) kernel
  trControl = train_control,
  tuneGrid = expand.grid(
    sigma = 1 / ncol(all_features_model_data),  # Equivalent to gamma in e1071
    C = 1                                     # Equivalent to cost
  )
)

# Print cross-validation results
print(svm_cv_model)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 2460 samples
##    9 predictor
##    2 classes: 'Loss', 'Win'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2214, 2214, 2214, 2214, 2214, 2214, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.6430894  0.2861789
##
## Tuning parameter 'sigma' was held constant at a value of 0.1
## Tuning
##  parameter 'C' was held constant at a value of 1
```

```
# Extract the mean accuracy across all folds
mean_accuracy <- mean(svm_cv_model$resample$Accuracy)
print(paste("Cross-Validated Testing Accuracy:", round(mean_accuracy * 100, 2), "%"))
```

```
## [1] "Cross-Validated Testing Accuracy: 64.31 %"
```

The SVM outperforms random forest given my features, so we will go with it.