final_project_654

Errol Kaylor

2022-12-09

Goal is to predict player ratings based on turns data!

Link to data: https://www.kaggle.com/competitions/scrabble-player-rating/data

Link to github repo: https://github.com/errolkaylor/654_edld_fp_scrabble

Project Intro

Our goal here is to understand how well we can predict player scrabble ratings based on online gameplay against computer opponents! Scrabble ratings are measures of relative scrabble skill of players, with a median score of tournament player being roughly 1200. Predicting player ratings is an important scrabble tournament problem, as creating fair divisons for competition is of the utmost importance. These models may also lead to simpler ELO calculations for players, as it is a relatively involved statistic to calculate for each player.

```
games <- import(here("data/games.csv/games.csv")) %>%
    as_tibble()
test <- import(here("data/test.csv/test.csv")) %>%
    as_tibble()
train <- import(here("data/train.csv/train.csv")) %>%
    as_tibble()
turns <- import(here("data/turns.csv/turns.csv")) %>%
    as_tibble()
#add player names to games data

games_train_full <- games %>%
    left_join(train) %>%
    filter(!(first==nickname)) %>%
    mutate(second_player=nickname) %>%
    separate(col=created_at,sep = " ",into = c("date","time")) %>%
    separate(col=date,sep="-",into = c("year","month","day")) %>%
    separate(col=time,sep=":",into = c("hour","minute","second")) %>%
    mutate_at(c(6:11),as.numeric)
```

```
## Joining, by = "game_id"
```

```
games_train_fuller <- games_train_full %>%
    mutate(game_winner = if_else(winner==1,second_player,first),
           winner_type = if_else(game_winner %in% c("BetterBot","STEEBot","HastyBot"),"Bot","Human")) %>
    select(-c(14,16,18)) %>%
    relocate(rating,.after = winner_type) %>%
    mutate(game_end_reason = as_factor(game_end_reason))

games_test_full <- games %>%
    left_join(test) %>%
    filter(!(first==nickname)) %>%
```

```
    mutate(second_player=nickname) %>%
    separate(col=created_at,sep = " ",into = c("date","time")) %>%
    separate(col=date,sep="-",into = c("year","month","day")) %>%
    separate(col=time,sep=":",into = c("hour","minute","second")) %>%
    mutate_at(c(6:11),as.numeric)
```

```
## Joining, by = "game_id"
```

```
games_test_fuller <- games_test_full %>%
    mutate(game_winner = if_else(winner==1,second_player,first),
           winner_type = if_else(game_winner %in% c("BetterBot","STEEBot","HastyBot"),"Bot","Human")) %>
    select(-c(14,16,18)) %>%
    relocate(rating,.after = winner_type) %>%
    mutate(game_end_reason = as_factor(game_end_reason)) %>%
    filter(!is.na(rating))

outcome <- c('rating')

id      <- c('game_id')

categorical <- c('time_control_name','game_end_reason','lexicon','rating_mode','winner_type')

numeric <- c('game_duration_seconds','score')

cyclic <- c('day','month','hour','minute','second')


blueprint_games <- recipe(x=games_train_fuller,
                          vars=c(id,outcome,categorical,numeric,cyclic),
                          roles=c('id','outcome',rep('predictor',12))) %>%
    step_harmonic('month',frequency = 1,cycle_size=3,role='predictor') %>%
    step_harmonic('day',frequency = 1,cycle_size=31,role='predictor') %>%
    step_harmonic('hour',frequency = 1,cycle_size=12,role='predictor') %>%
    step_harmonic('minute',frequency = 1,cycle_size=60,role='predictor') %>%
    step_harmonic('second',frequency = 1,cycle_size=60,role='predictor') %>%
    step_dummy('lexicon',one_hot = TRUE) %>%
    step_dummy('time_control_name',one_hot=TRUE) %>%
    step_dummy('winner_type',one_hot=TRUE) %>%
    step_dummy('rating_mode',one_hot=TRUE) %>%
    step_dummy('game_end_reason',one_hot=TRUE) %>%
    step_normalize('game_duration_seconds') %>%
    step_normalize('score') %>%
    step_normalize(c('month_sin_1','day_sin_1','hour_sin_1','minute_sin_1','second_sin_1','month_cos_1'
```

Our dataset consists of game, player rating, and turn data for ~73,000 scrabble games played against bots on woogles.io. For game, we are given a game id, player names, which player won and went first, game creation time and date, and the dictionary ("Lexicon") being used in the particular game. Additionally, we are given test and training datasets with game ids, player scores, and player ratings before the game is played. Finally, we have turn data for each scrabble game played, that includes infomration on current player score, player move type, player points scored (if points were scored with the move), where the player put their tiles, and what tiles they have in their rack. I used player rack to calculate a new metric, rack score. This metric is the total point value of tiles in the players rack before each move, and is used in comparison to points scored each turn to provide more evidence for scrabble skill.

```r
games_train_fuller = games_train_fuller[sample(nrow(games_train_fuller)),]

folds1 = cut(seq(1,nrow(games_train_fuller)),breaks=10,labels=FALSE)

# Create the list for each fold

my.indices1 <- vector('list',10)

for(i in 1:10){
    my.indices1[[i]] <- which(folds1!=i)
}
cv <- trainControl(method          = "cv",
                   index           = my.indices1)


ff_glimpse(games_train_fuller)
```

```
## $Continuous
##                                      label var_type     n missing_n
## game_id                            game_id    <int> 50410         0
## winner                              winner    <int> 50410         0
## year                                  year    <dbl> 50410         0
## month                                month    <dbl> 50410         0
## day                                    day    <dbl> 50410         0
## hour                                  hour    <dbl> 50410         0
## minute                              minute    <dbl> 50410         0
## second                              second    <dbl> 50410         0
## initial_time_seconds   initial_time_seconds    <int> 50410         0
## game_duration_seconds game_duration_seconds    <dbl> 50410         0
## score                                score    <int> 50410         0
## rating                              rating    <int> 50410         0
##                         missing_percent    mean      sd    min quartile_25
## game_id                             0.0 36367.3 21020.4    1.0     18139.8
## winner                              0.0     0.4     0.5   -1.0         0.0
## year                                0.0  2022.0     0.0 2022.0      2022.0
## month                               0.0     8.5     0.6    7.0         8.0
## day                                 0.0    15.3     9.0    1.0         7.0
## hour                                0.0    11.3     6.9    0.0         5.0
## minute                              0.0    29.5    17.2    0.0        15.0
## second                              0.0    29.5    17.3    0.0        14.0
## initial_time_seconds                0.0  1216.8   713.0   15.0       900.0
## game_duration_seconds               0.0   492.5   330.6   19.8       260.8
## score                               0.0   392.6    74.8  -64.0       345.0
## rating                              0.0  1876.5   232.5 1033.0      1664.0
##                          median quartile_75     max
## game_id                 36339.5     54624.8 72773.0
## winner                      0.0         1.0     1.0
## year                     2022.0      2022.0  2022.0
## month                       9.0         9.0     9.0
## day                        15.0        22.0    31.0
## hour                       12.0        17.0    23.0
## minute                     29.0        44.0    59.0
## second                     29.0        44.0    59.0
## initial_time_seconds     1200.0      1200.0  3600.0
```

```
## game_duration_seconds   411.5       638.5  4444.8
## score                   391.5       439.0   939.0
## rating                 1909.0      2062.0  2510.0
##
## $Categorical
##                              label var_type     n missing_n missing_percent
## first                        first    <chr> 50410         0             0.0
## time_control_name time_control_name    <chr> 50410         0             0.0
## game_end_reason     game_end_reason    <fct> 50410         0             0.0
## lexicon                    lexicon    <chr> 50410         0             0.0
## rating_mode            rating_mode    <chr> 50410         0             0.0
## second_player        second_player    <chr> 50410         0             0.0
## game_winner            game_winner    <chr> 50410         0             0.0
## winner_type            winner_type    <chr> 50410         0             0.0
##                    levels_n                                              levels
## first                   915                                                   -
## time_control_name         4                                                   -
## game_end_reason           4 "STANDARD", "RESIGNED", "TIME", "CONSECUTIVE_ZEROES"
## lexicon                   4                                                   -
## rating_mode               2                                                   -
## second_player           931                                                   -
## game_winner             896                                                   -
## winner_type               2                                                   -
##                              levels_count             levels_percent
## first                                   -                          -
## time_control_name                       -                          -
## game_end_reason   47694, 1284, 1249, 183 94.61,  2.55,  2.48,  0.36
## lexicon                                 -                          -
## rating_mode                             -                          -
## second_player                           -                          -
## game_winner                             -                          -
## winner_type                             -                          -
```

```
mod_1 <- caret::train(blueprint_games,
                      data=games_train_fuller,
                      method='glmnet',
                      tuneGrid = expand.grid( alpha = seq(0.0001,1,length = 20),
                                              lambda = seq(0.0001,1,length = 20)),
                      trControl = cv)
```

```
## Loading required namespace: glmnet
```

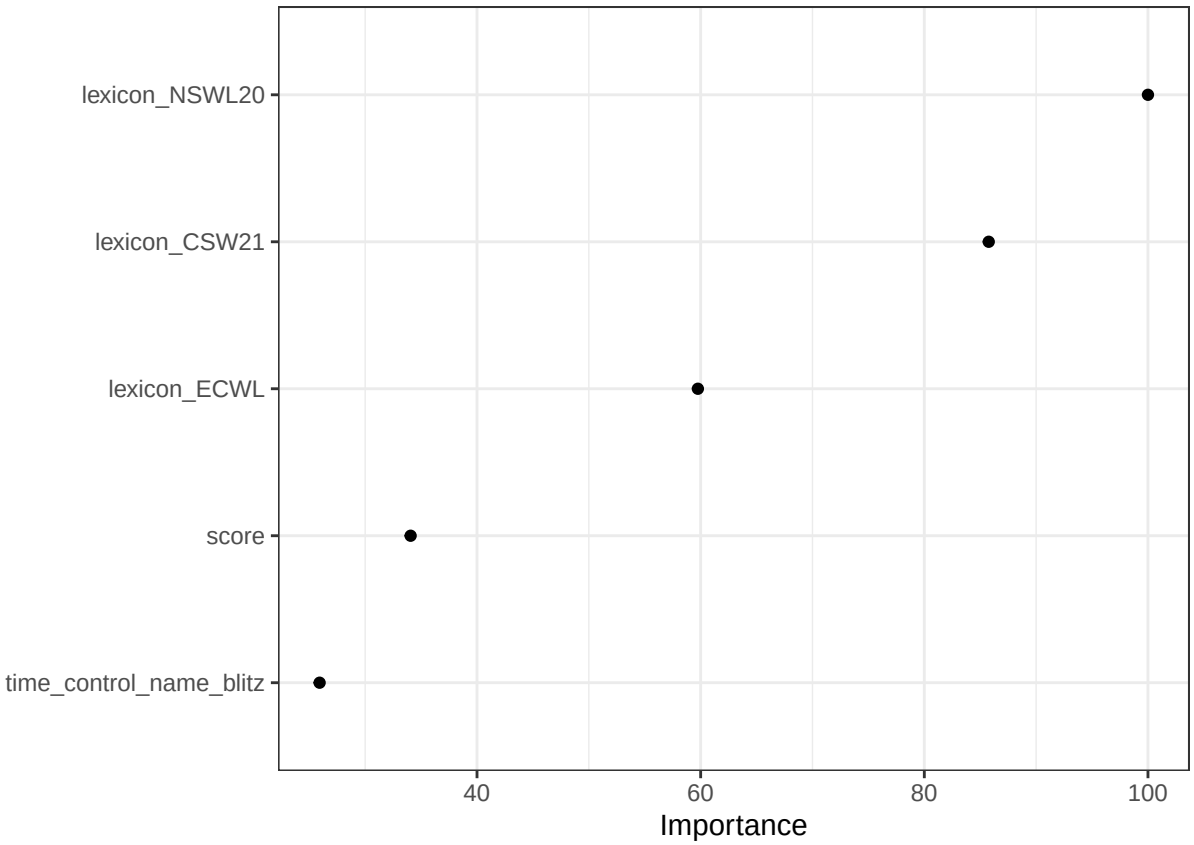
```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-4
```

```
#mod_1

vip(mod_1,num_features = 5, geom = "point") +
    theme_bw()
```




```
rack_val <- function(rack_string){
                one <- str_count(rack_string,"[AEIOULNSTR]")
                two <- str_count(rack_string,"[DG]")*2
                three <- str_count(rack_string,"[BCMP]")*3
                four <-  str_count(rack_string,"[FHVWY]")*4
                five <-  str_count(rack_string,"[K]")*5
                six <-  str_count(rack_string,"[JX]")*8
                seven <- str_count(rack_string,"[QZ]")*10
                sum <- one + two + three + four + five + six + seven
}

turns <- turns %>%
    mutate(rack_values = rack_val(rack),
           proportion_points_scored = points/rack_values)

turns_summary <- turns %>%
    group_by(game_id) %>%
    summarize(avg_prop_points_scored = mean(proportion_points_scored,na.rm=TRUE),
              avg_points_scored = mean(points,na.rm=TRUE))

games_turns_combo <- games_train_fuller %>%
    left_join(turns_summary) %>%
    relocate(rating,.after = avg_points_scored) %>%
    filter(is.finite(avg_prop_points_scored ))
```

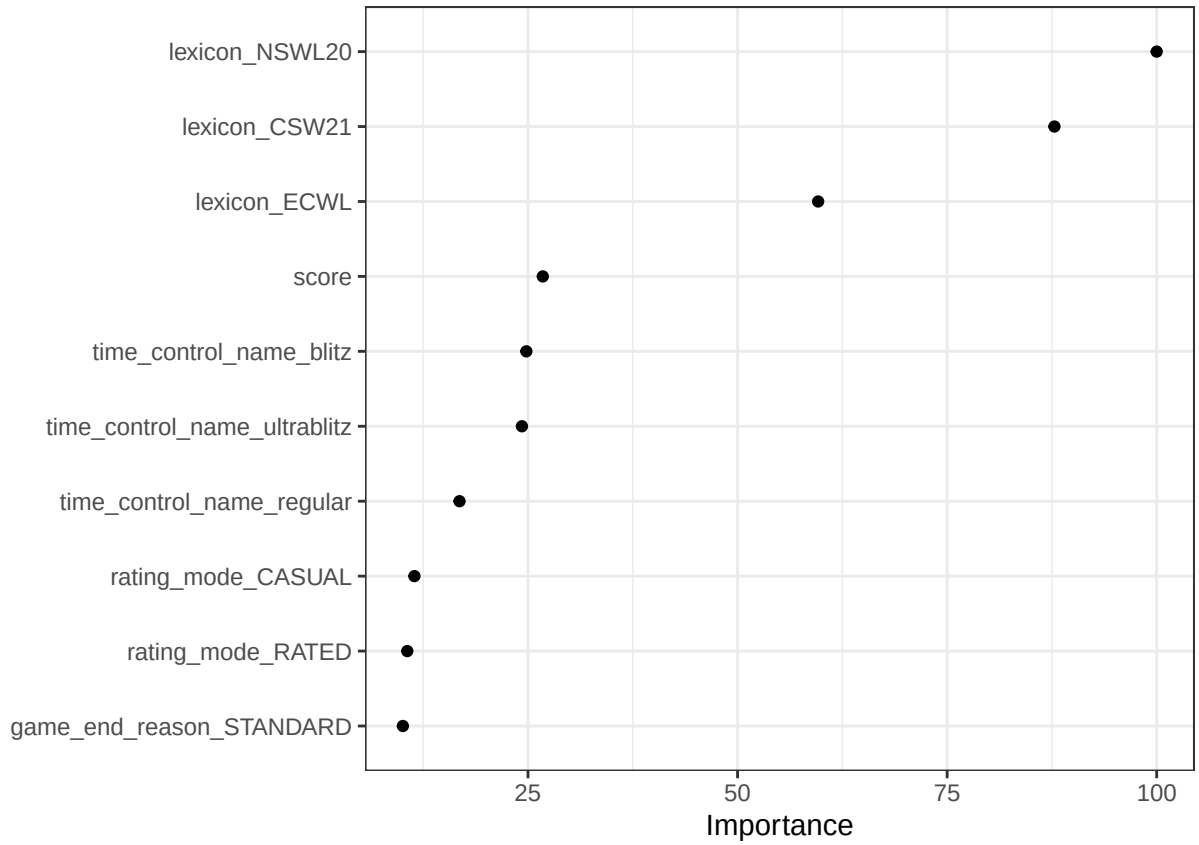
```
## Joining, by = "game_id"
```

```
games_test_turns_combo <- games_test_fuller %>%
    left_join(turns_summary) %>%
    relocate(rating,.after = avg_points_scored) %>%
    filter(is.finite(avg_prop_points_scored ))
```

```
## Joining, by = "game_id"
```

```
numeric_updated <- c('game_duration_seconds','score','avg_prop_points_scored','avg_points_scored')


blueprint_games_turns_combo <- recipe(x=games_turns_combo,
                          vars=c(id,outcome,categorical,numeric_updated,cyclic),
                          roles=c('id','outcome',rep('predictor',14))) %>%
    step_harmonic('month',frequency = 1,cycle_size=3,role='predictor') %>%
    step_harmonic('day',frequency = 1,cycle_size=31,role='predictor') %>%
    step_harmonic('hour',frequency = 1,cycle_size=12,role='predictor') %>%
    step_harmonic('minute',frequency = 1,cycle_size=60,role='predictor') %>%
    step_harmonic('second',frequency = 1,cycle_size=60,role='predictor') %>%
    step_dummy('lexicon',one_hot = TRUE) %>%
    step_dummy('time_control_name',one_hot=TRUE) %>%
    step_dummy('winner_type',one_hot=TRUE) %>%
    step_dummy('rating_mode',one_hot=TRUE) %>%
    step_dummy('game_end_reason',one_hot=TRUE) %>%
    step_normalize('game_duration_seconds') %>%
    step_normalize('score') %>%
    step_normalize('avg_points_scored') %>%
    step_normalize('avg_prop_points_scored') %>%
    step_normalize(c('month_sin_1','day_sin_1','hour_sin_1','minute_sin_1','second_sin_1','month_cos_1'

games_turns_combo = games_turns_combo[sample(nrow(games_turns_combo)),]

folds2 = cut(seq(1,nrow(games_turns_combo)),breaks=10,labels=FALSE)

# Create the list for each fold

my.indices2 <- vector('list',10)

for(i in 1:10){
    my.indices2[[i]] <- which(folds2!=i)
}
cv <- trainControl(method          = "cv",
                   index           = my.indices2)


mod_2 <- caret::train(blueprint_games_turns_combo,
                      data=games_turns_combo,
                      method='glmnet',
                      tuneGrid = expand.grid( alpha = seq(0.0001,1,length = 20),
                                              lambda = seq(0.0001,1,length = 20)),
                      trControl = cv)

#mod_2

vip(mod_2,num_features = 10, geom = "point") +
```

```
    theme_bw()
```



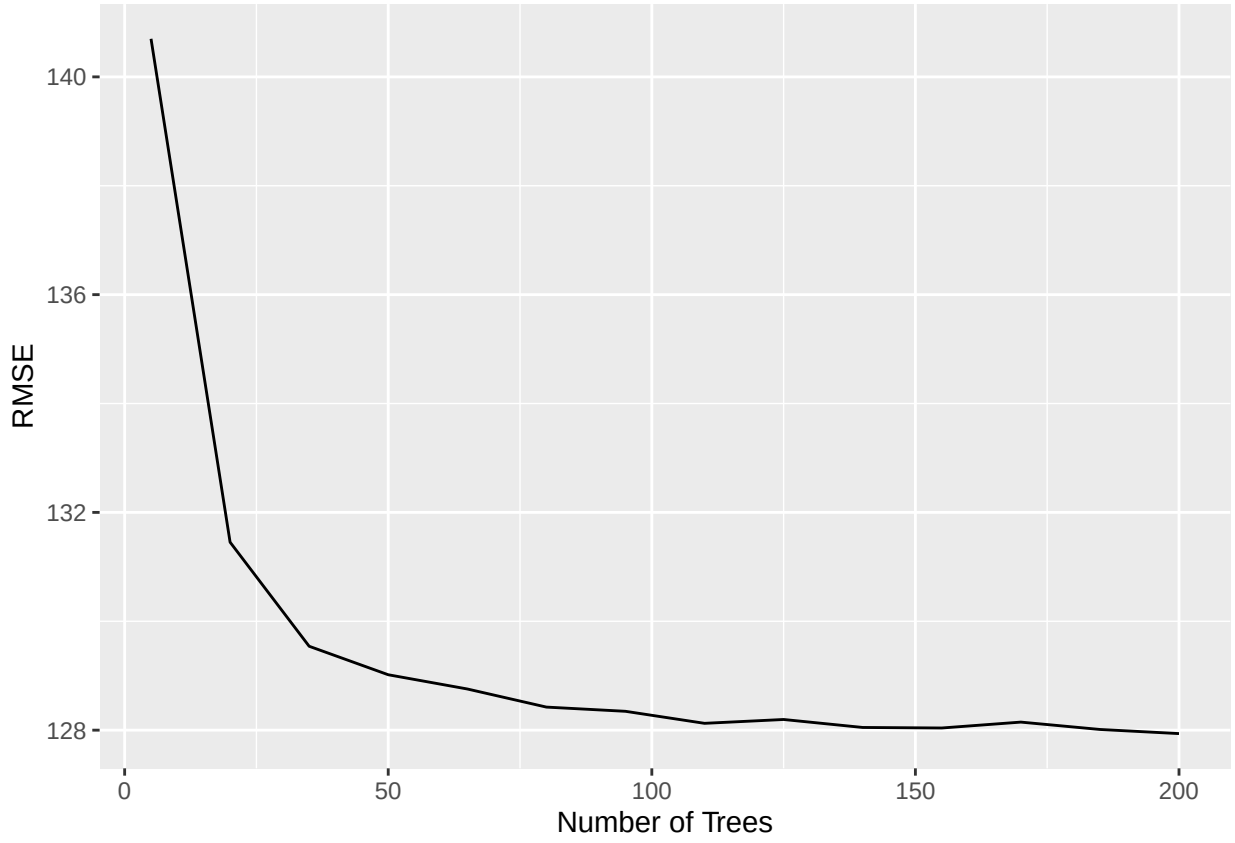

```
rangergrid <- expand.grid(
    .mtry=10,
    .splitrule='variance',
    .min.node.size=2)

nbags <- c(seq(5,200,15))

bags<- vector('list',length(nbags))

for(i in 1:length(nbags)){
    bags[[i]] <- caret::train(blueprint_games_turns_combo,
                              data = games_turns_combo,
                              method = 'ranger',
                              tuneGrid = rangergrid,
                              trControl = cv,
                              num.trees = nbags[i],
                              importance = 'impurity',
                              max.depth = 50)
}
```

```
## Loading required namespace: e1071
```

```
## Loading required namespace: ranger
```

```
## Growing trees.. Progress: 94%. Estimated remaining time: 45 seconds.
```

```
rmses <- c()
for(i in 1:length(nbags)){

  rmses[i] = bags[[i]]$results$RMSE

}
ggplot()+
  geom_line(aes(x=nbags,y=rmses))+
  xlab('Number of Trees')+
  ylab('RMSE')
```




```
nbags[which.min(rmses)]
```

```
## [1] 200
```

```
predicted_te_mod1 <- predict(mod_1,games_test_turns_combo)

predicted_te_mod2 <- predict(mod_2,games_test_turns_combo)



predicted_te <- predict(bags[[14]],games_test_turns_combo)


# MAE
```

```
rf_mae <- mean(abs(games_test_turns_combo$rating - predicted_te))
# RMSE
rf_rmse <- sqrt(mean((games_test_turns_combo$rating - predicted_te)^2))
# R-square
rf_rsq <- cor(games_test_turns_combo$rating,predicted_te)^2

# MAE
mod1_mae <- mean(abs(games_test_turns_combo$rating - predicted_te_mod1))
# RMSE
mod1_rmse <- sqrt(mean((games_test_turns_combo$rating - predicted_te_mod1)^2))
# R-square
mod1_rsq <- cor(games_test_turns_combo$rating,predicted_te_mod1)^2

# MAE
mod2_mae <- mean(abs(games_test_turns_combo$rating - predicted_te_mod2))
# RMSE
mod2_rmse <- sqrt(mean((games_test_turns_combo$rating - predicted_te_mod2)^2))
# R-square
mod2_rsq <- cor(games_test_turns_combo$rating,predicted_te_mod2)^2



agg_perf <- c("Pure Game Data","Added Turn Data","Random Forest") %>%
    as_tibble() %>%
    mutate(rsq = c(mod1_rsq,mod2_rsq,rf_rsq),
           rmse = c(mod1_rmse,mod2_rmse,rf_rmse),
           mae = c(mod1_mae,mod2_mae,rf_mae))

kable(agg_perf,digits = 3)
```

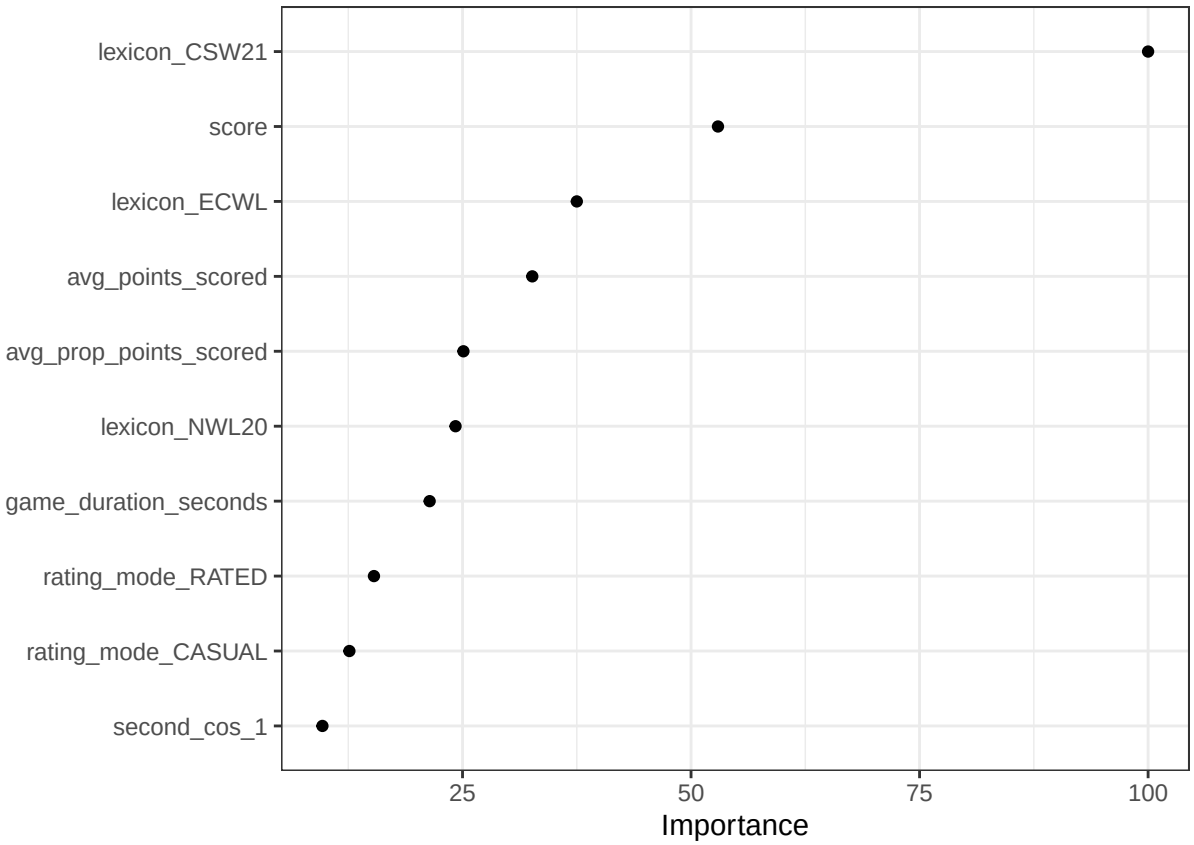
value	rsq	rmse	mae
Pure Game Data	0.546	136.696	110.773
Added Turn Data	0.590	133.131	107.559
Random Forest	0.647	119.667	95.969

I ran a series of three models on our scrabble data, being two elasticnet penalized regressions (varying both our alpha and lambda values from 0-1 for optimizing our model) and a random forest model using the ranger function. Our initial model is predicting player ratings without using any turn metadata, and performs poorly as a result, with rsq = .546, RMSE = 136.7, and MAE = 110.773.

Including turn metadata slightly improved the performance of our model as we can see with model 2 but not significantly. Our rsq rises and RMSE and MAE drop slightly respectively, with values of .59, 133.15, and 107.57 respectively.

The final random forest model outperformed both our elasticnet regressions, with rsq = .648, RMSE = 119.95, and MAE = 96.36

```
vip(bags[[14]],num_features = 10, geom = "point") +
    theme_bw()
```



From an intial feature analysis, our random forest model is aso the most promising, as it values features that make intuitive sense - our average points scored, average proportion of points scored, player score, and how long the game took.