League of Legends and Naive Bayes

Yuhan (Emma) Zhang

2025-05-20

Research question and explore the data

As a League of Legends player, I like to analyze players' behavior and demographics.

One day, I met a new friend named Alex, but I wasn't sure whether she plays League of Legends. Fortunately, I have some information about her gender, age, gaming habits, and preferred game genres.

Alex:

1. has played video games in the past few week
2. is a women, pronouns are she/her
3. birth year 2000
4. has average gaming hours per week 30 hours
5. is currently in a relationship
6. never goes out
7. plays Rocket League
8. plays Valorant
9. plays Fortnite
10. plays Minecraft
11. plays Genshin Impact
12. plays The Sims

Our research question is: Can we accurately predict whether a person (like Alex) plays League of Legends using demographic and gaming behavior features: player status, gender, age, gaming hours, going out frequency, preferred games, and romantic status?

Our goal is to use Naive Bayes and random forest (two supervised learning methods) to answer this question. First of all, we need to install our data and some specific library sets:

```
library(tidyverse)
library(dplyr)
library(readr)
library(janitor)
```

```r
video_game_data <- read_csv("video_games.csv")

video_game_data <- video_game_data |>
  janitor::clean_names()
```

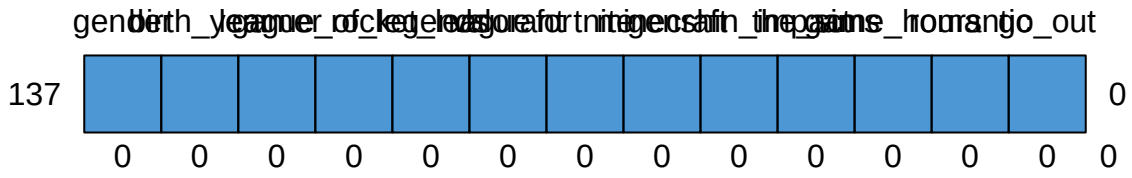
The data that we are going to use to answer our research question is "video_game_data". This is a data set collected through a questionnaire distributed on Discord groups. We'll need to understand the data first. The variables in this data set are:

1. gender
2. birth year
3. gamer: if the respondent has played video games in the past few week
4. League of Legends : yes or no
5. Rocket League : yes or no
6. Valorant : yes or no
7. Fortnite : yes or no
8. Minecraft : yes or no
9. Genshin Impact : yes or no
10. The Sims : yes or no
11. game hours : less than 1 hour, 1-5 hours, 5-20 hours, 20-50 hours, more than 50 hours
12. romantic : single or in a relationship
13. go out : on a scale from 1 to 4 (1 = never going out, 4 = always going out)

Let's check if our data set has missing data using "mice" package.

```r
library(mice)
md.pattern(video_game_data)
```

```
##  /\     /\
## {  `---'  }
## {  0   0  }
## ==>  V <==  No need for mice. This data set is completely observed.
##  \  \|/  /
##   `-----'
```




```
##     gender birth_year gamer league_of_legends rocket_league valorant fortnite
## 137      1          1     1                 1             1        1        1
##          0          0     0                 0             0        0        0
##     minecraft genshin_impact the_sims game_hours romantic go_out
## 137         1              1        1          1        1      1 0
##             0              0        0          0        0      0 0
```

There are no missing value. Good! We can continue our research.

Naive Bayes and its Explanation

I'll explain what the Naive Bayes (Multinomial Naive Bayes and Gaussian Naive Bayes) are in this section, and we'll go through the Naive Bayes calculation step by step.

1. Make an initial guess if Alex plays League of Legends.

This guess is called the Prior Probabilities.

For our data set, 33 people play League of Legends, and 104 people don't. Thus, our prior probabilities are:

```
P(Plays League of Legends) = 33/134 = 0.2409

P(Doesn't Play League of Legends) = 104/134 = 0.7591
```

2. Calculate the likelihood that Alex is a women given that she plays LOL. Then, calculate the likelihood that Alex is a woman given that she does not play LOL.

There are 6 out of 33 people who play LOL declare that they are women. So, to calculate the likelihood that Alex is a women given that she plays LOL, we use this formula:

```
P(female | plays LOL) = 6/33 = 0.1818182
```

There's 57 out of 104 people who do not play LOL declare that they are women. Similarly, to calculate the likelihood that Alex is a women given that she does not play LOL, we use this formula:

```
P(female | does not play LOL) = 57/104 = 0.5480769
```

3. Repeat step 2 for every variable left: player status, romantic status, birth year, rocket league, valorant, fortnite, minecraft, genshin impact, the sims, go out, and gaming hours.

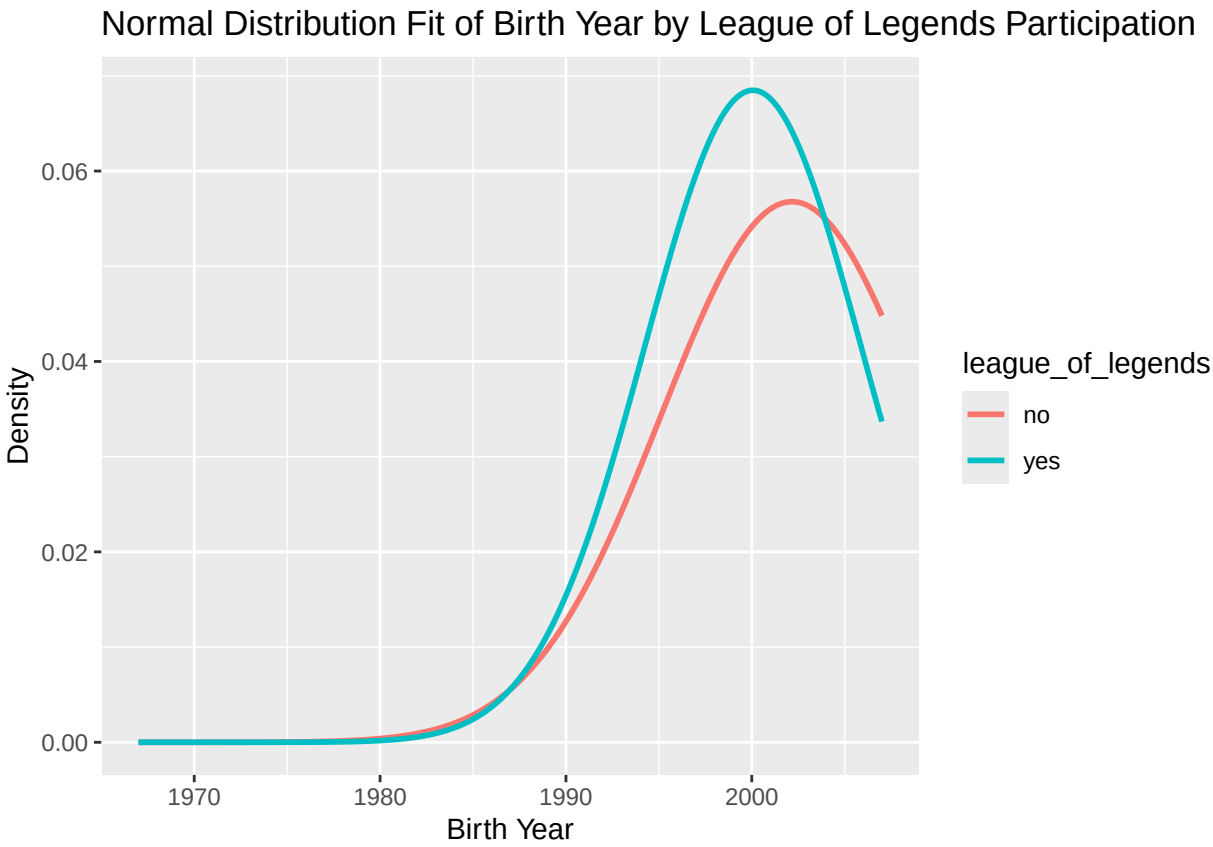
There are called Categorical (Multinomial) Naive Bayes, the calculation for categorical variables.

```
P(is a gamer | plays LOL) = 33/33 = 1

P(is a gamer | does not play LOL) = 83/104 = 0.7980769

P(in a relationship | plays LOL) = 14/33 = 0.4242424

P(in a relationship | does not play LOL) = 34/104 = 0.3269231

P(plays 20-50 hours | plays LOL) = 14/33 = 0.4242424

P(plays 20-50 hours | does not play LOL) = 10/104 = 0.09615385

P(plays fortnite | does play LOL) = 0.455

P(plays fortnite | does not play LOL) = 0.24

...
```

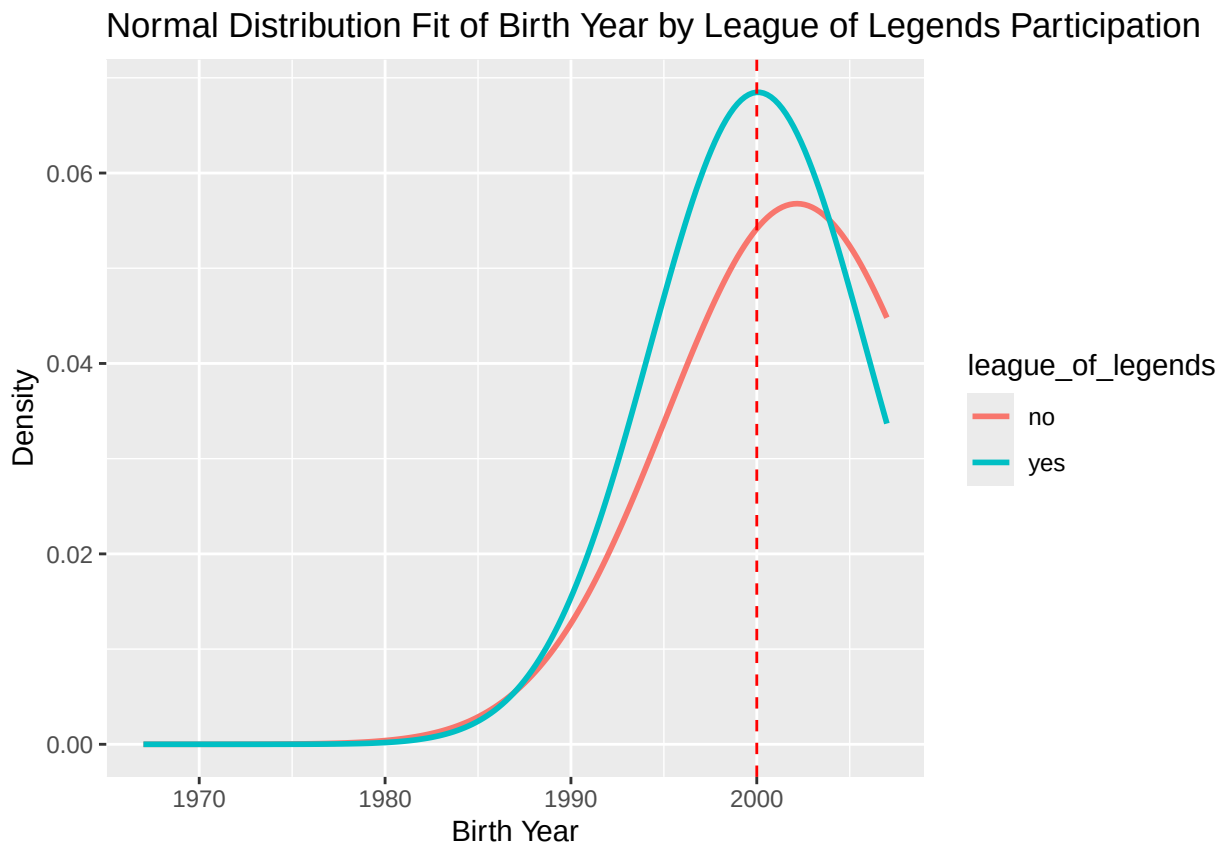
What if the variable is a numerical one, like birth year?

Well, that's where Gaussian Naive Bayes comes in.

We fit a Gaussian (or Normal) distribution using the mean and standard deviation of the birth year for people who play League of Legends and those who do not. The graph looks like the one below.



Remember that Alex was born in the year 2000. So we draw a vertical line at 2000 in this graph. The line intersect with the two normal distributions, and the two corresponding y values are the likelihood that Alex was born in the year 2000 given that she plays LOL and she does not play LOL.




```
P(born in 2000 | plays LOL) = 0.068

P(born in 2000 | does not play LOL) = 0.055
```

NOTE: In most cases, year (like birth_year) is treated as a numerical (continuous or discrete) variable. However, you could treat year as a categorical variable if you have very few unique years and each year represents a meaningful group. In this report, I treat born year as a numerical variable because I want to show you what GNB would do if the variable is a numerical variable. Also, later when we build the naiveBayes model using e1071 package, if a column is numeric (e.g., 1985, 2000), e1071's naiveBayes will automatically treat it as numerical variable. So, go_out variable is also a numerical variable in naiveBayes model. We might want to fix this in the future, but for now, treating it as a numerical variable causes no so much problem.

While this graph may not look so normal, the naiveBayes model would treat it as a normal distribution.

4. Now we get the likelihoods. What do we do next? Well, we multiply the Prior Probabilities and all the likelihoods together for both LOL players and non-LOL players:

```
P(Plays League of Legends) * P(female | plays LOL) * P(is a gamer | plays LOL) *
P(in a relationship | plays LOL) * P(plays 20-50 hours | plays LOL) *
P(born in 2000 | plays LOL) * P(plays fortnite | does play LOL) *
```

```
P(plays minecraft | does play LOL) * P(plays rocket league | does play LOL) *
P(plays valorant | does play LOL) * P(plays genshin impact | does play LOL) *
P(plays the sims | does play LOL) * P(go out = 1 | does play LOL)= 9.90 × 10^-9

P(Does not play League of Legends) * P(female | does not play LOL) *
P(is a gamer | does not play LOL) * P(in a relationship | does not play LOL) *
P(plays 20-50 hours | does not play LOL) * P(born in 2000 | does not play LOL) *
P(plays fortnite | does not play LOL) * P(plays minecraft | does not play LOL) *
P(plays rocket league | does not play LOL) * P(plays valorant | does not play LOL) *
P(plays genshin impact | does not play LOL) * P(plays the sims | does not play LOL) *
P(go out = 1 | does not play LOL)
= 1.70 × 10^-9
```

The number that we get for Alex is a LOL player is greater than the number we get for Alex is not a LOL player. Thus, we will classify Alex as a LOL player.

ANOTHER NOTE: We are getting really really small numbers. This is the time where we have to prevent Underflow. What is Underflow?

Underflow occurs in computing when you perform calculations with very small numbers — so small that the computer rounds them to zero because it can't represent them accurately with its limited number of decimal places.

How to prevent Underflow? Well, instead of multiplying the Ps directly, we take the log() of every number and add them together. The number will change, but when comparing the two numbers, the bigger score for plays LOL will stay big.

Address the Research Question using Naive Bayes

In R Studio, we don't have to do the above steps. The computer is going to do all of these for us by using the e1071 package.

```
library(e1071)
```

We first use this package to build a Naive Bayes model. This model can handle both continuous and categorical data. It uses Gaussian Naive Bayes for numeric features, and uses Categorical (Multinomial) Naive Bayes for categorical variables.

So it's actually doing a hybrid model of Gaussian Naive Bayes and Multinomial Naive Bayes, making it highly effective.

```
video_game_data$gamer <- factor(video_game_data$gamer)
video_game_data$gender <- factor(video_game_data$gender)
video_game_data$game_hours <- factor(video_game_data$game_hours)
video_game_data$romantic <- factor(video_game_data$romantic)
video_game_data$the_sims <- factor(video_game_data$the_sims)
video_game_data$rocket_league <- factor(video_game_data$rocket_league)
video_game_data$fortnite <- factor(video_game_data$fortnite)
video_game_data$minecraft <- factor(video_game_data$minecraft)
video_game_data$valorant <- factor(video_game_data$valorant)
```

```
video_game_data$genshin_impact <- factor(video_game_data$genshin_impact)
#Don't forget to as.factor the categorical variables in the data set!

gnb_model <- naiveBayes(factor(league_of_legends) ~ gamer + gender + game_hours
                        + romantic + birth_year + rocket_league + valorant +
                          fortnite + minecraft + genshin_impact + the_sims + go_out,
                        data = video_game_data,
                        laplace = 1)
#you can avoid zero probabilities (which cause prediction issues) using the laplace = 1 argument.
```

In the above chunk, the code did all the steps we learned in the previous section for this data set.

This model is ready to go! Now all you need is to use predict() and fit Alex's data into it!

```
alex_player <- data.frame(
  gamer = factor("yes", levels = levels(video_game_data$gamer)),
  gender = factor("Woman", levels = levels(video_game_data$gender)),
  game_hours = factor("20-50 hours", levels = levels(video_game_data$game_hours)),
  romantic = factor("In a relationship", levels = levels(video_game_data$romantic)),
  birth_year = 2000,
  rocket_league = factor("yes", levels = levels(video_game_data$rocket_league)),
  the_sims = factor("yes", levels = levels(video_game_data$the_sims)),
  valorant = factor("yes", levels = levels(video_game_data$valorant)),
  fortnite = factor("yes", levels = levels(video_game_data$fortnite)),
  minecraft = factor("yes", levels = levels(video_game_data$minecraft)),
  genshin_impact = factor("yes", levels = levels(video_game_data$genshin_impact)),
  go_out = 1
)
```

```
alex_predict_gnb <- predict(gnb_model, alex_player)
print(alex_predict_gnb)
```

```
## [1] yes
## Levels: no yes
```

The Naive Bayes model in e1071 package also says that Alex is a LOL player. This proves that our calculation is correct. This model could predict whether a person plays League of Legends using demographic and gaming behavior features: player status, gender, age, gaming hours, romantic status, going out frequency, and preferred games, so it addresses our research problem.

But how well does this model do? Well, let's split the data in two and use one half to build a Naive Bayes model, and use that model on the other half.

```
set.seed(10000)
training_data_rows <- sample(1:nrow(video_game_data), size = nrow(video_game_data)/2)

training_data <- video_game_data[training_data_rows, ]
testing_data <- video_game_data[-training_data_rows, ]

gnb_model_training <- naiveBayes(factor(league_of_legends) ~ gamer + gender + game_hours
                        + romantic + birth_year + rocket_league + valorant +
                          fortnite + minecraft + genshin_impact + the_sims + go_out,
                        data = training_data,
```

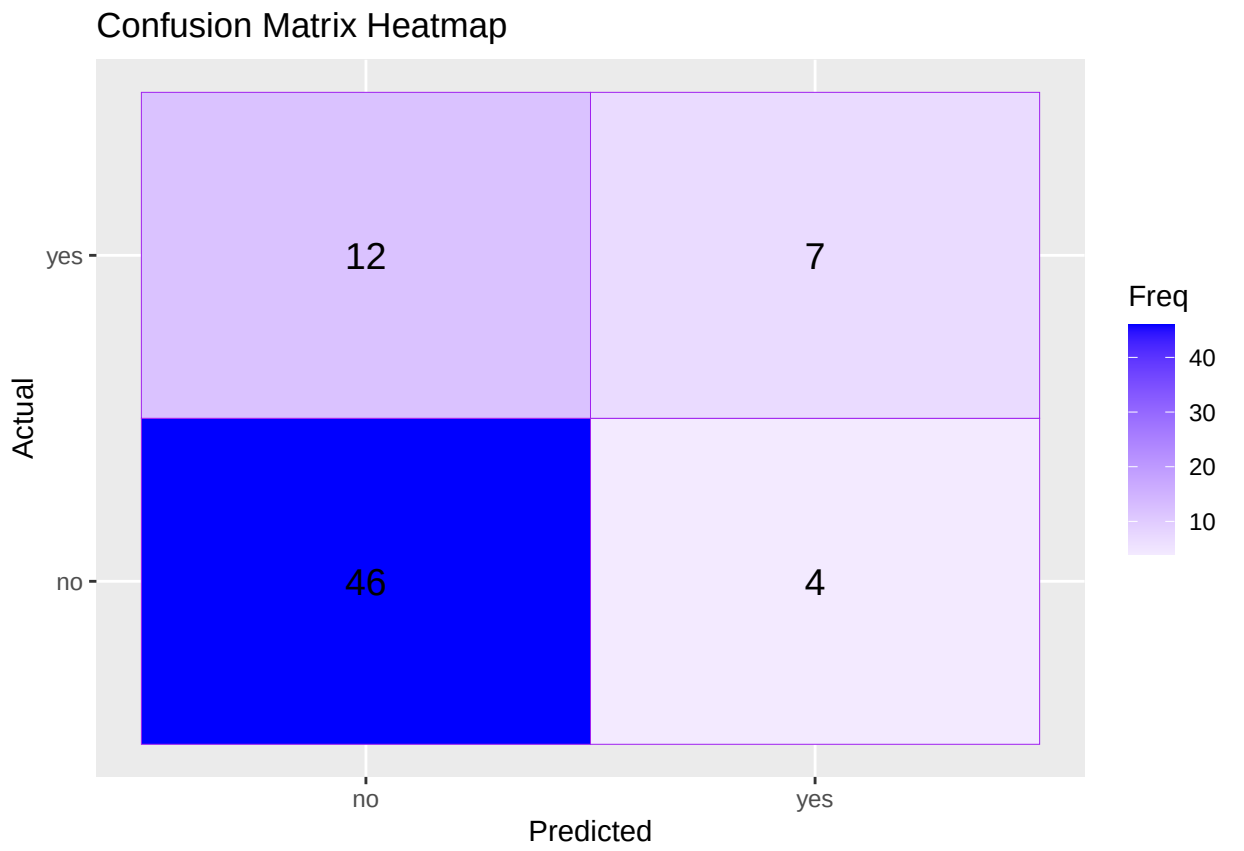
```
                        laplace = 1)

predictions_gnb_test <- predict(gnb_model_training, newdata = testing_data)
```

The confusion matrix is as below:

```
conf_matrix <- table(Predicted = predictions_gnb_test,
                     Actual = testing_data$league_of_legends)

print(conf_matrix)
```

```
##          Actual
## Predicted no yes
##       no  46  12
##       yes  4   7
```

```
conf_df <- as.data.frame(conf_matrix)

ggplot(data = conf_df, aes(x = Predicted, y = Actual, fill = Freq)) +
  geom_tile(color = "purple") +
  geom_text(aes(label = Freq), size = 5) +
  scale_fill_gradient2(low = "red", high = "blue", mid = "white") +
  labs(title = "Confusion Matrix Heatmap",
       x = "Predicted",
       y = "Actual")
```



It looks like the model is doing very good! It gets almost every person's LOL status, only 16 predictions are wrong! This means that the accuracy for our NB model is 53/69 = 0.7681159.

What are some strength and drawbacks of Naive Bayes?

Strength:

1. naiveBayes in e1071 could handle Mixed Data Types. It can process both numerical (continuous) and categorical (factor) variables in the same model. Numerical variables are modeled using a Gaussian distribution. Categorical variables use Multinomial Naive Bayes.

Weakness:

1. Strong Independence Assumption. Naive Bayes assumes that all features are conditionally independent given the class label. But in reality, features like gamer and LOL status are often correlated.
2. Assumes Normal (Gaussian) Distribution of Features Gaussian NB assumes that each continuous feature follows a normal distribution within each class. But real data often have skewed, bimodal, or otherwise non-normal distributions. This mismatch can reduce the quality of probability estimates and predictions.

Address the Research Question using Random Forest

Let's use random forest model to address the question again, just to make sure.

```
library(caret)

model_rf <- train(factor(league_of_legends) ~ gamer + gender + game_hours
                        + romantic + birth_year + rocket_league + valorant +
                          fortnite + minecraft + genshin_impact + the_sims + go_out,
                        data = video_game_data,
                        method = "rf")

alex_prediction_rf <- predict(model_rf, alex_player)
print(alex_prediction_rf)
```

```
## [1] yes
## Levels: no yes
```

Random forest tells the same story with our NB model. Random Forest says that Alex is a LOL-player. Let's check random forest's accuracy.

```
model_rf
```

```
## Random Forest
##
## 137 samples
```

```
##  12 predictor
##   2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 137, 137, 137, 137, 137, 137, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.8436510  0.5141475
##    9    0.8489745  0.5771864
##   16    0.8384219  0.5505587
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 9.
```

It looks like mtry = 9 has the greatest accuracy and kappa. Let me fix my code:

```
model_rf_2 <- train(factor(league_of_legends) ~ gamer + gender + game_hours
                        + romantic + birth_year + rocket_league + valorant +
                          fortnite + minecraft + genshin_impact + the_sims + go_out,
                      data = video_game_data,
                      method = "rf",
                      tuneGrid = data.frame(mtry = c(9, 10, 11, 12, 13, 14, 15, 16)))
```

Let's look at the new rf model.

```
model_rf_2
```

```
## Random Forest
##
## 137 samples
##  12 predictor
##   2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 137, 137, 137, 137, 137, 137, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    9    0.8616321  0.5959001
##   10    0.8664785  0.6057468
##   11    0.8614435  0.5993327
##   12    0.8543830  0.5776649
##   13    0.8527274  0.5733446
##   14    0.8536838  0.5742056
##   15    0.8470777  0.5563425
##   16    0.8489155  0.5573707
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 10.
```

Among the values tested, mtry = 10 yielded the highest accuracy (86.6%) and the highest Kappa (0.6057), indicating that this model closed 60.57% of the gap between random guessing and calculating. It does have a slight higher accuracy than our NB model (approximately by 11%), but it is not to be concluded that it is better than NB. While testing NB model, we could see the percentage of false negative and false positive, which might be helpful. In contrast, while Random Forest often achieves higher predictive performance, it functions more like a black box, making it harder to interpret the influence of individual predictors.

Also, we have a small data set with many categorical variables, which is not the best environment for random forest models. With so few rows, especially if some features are categorical with many levels, Random Forest can memorize training data instead of generalizing. Naive Bayes would do good in small sample size and mostly categorical variables because Naive Bayes relies on estimating simple probabilities, which are relatively robust and don't overfit.

Despite all, this RF model also address our research model. Using "alex_model_rf_2", we could predict whether a person plays League of Legends using demographic and gaming behavior features: player status, gender, age, gaming hours, and romantic status.

Research question solved!