Exercise 3

2024-01-29

Description of your predictive model

The predictive model is an XGBoost classifier aimed at predicting examiner turnover at the USPTO. It uses a select set of features, such as the number of new, abandoned, and issued applications, as well as the composition of the examiner's art unit. Trained with 5-fold cross-validation, the model achieved an AUC of 0.940, indicating a strong capability to differentiate between examiners who are likely to leave and those who will stay

The XGBoost classifier functions by employing gradient boosting algorithms to construct an ensemble of decision trees, iteratively refining predictions through successive training rounds. Each tree in the ensemble is built to address and correct the errors of the preceding one, thus enhancing the model's accuracy

Utilizing a subset of available variables, the model predicts the separation_indicator, a binary variable signifying whether an examiner is likely to leave their position. The features include:

num_new_applications: The count of new patent applications assigned to an examiner. num_abandoned_applications: The count of applications abandoned during the examination process. num_issued_patents: The count of patents issued under an examiner's review. num_people_in_art_unit: The number of examiners in the same art unit. num_women_in_art_unit: The number of female examiners in the same art unit.

Interpret your model

A 5-fold cross-validation method was employed during the model training phase to mitigate overfitting and ensure robustness. The model's performance was primarily evaluated using the Area Under the Receiver Operating Characteristic (ROC) Curve (AUC), with the final model achieving an AUC of approximately 0.940 on the test set, suggesting a strong predictive capability.

The XGBoost model exhibits a high level of discrimination between potential outcomes, as indicated by the ROC curve and the accompanying AUC metric. The convergence of AUC values between the training and test sets illustrates the model's generalization ability, confirming that the performance is not merely an artifact of the training data.

Application of the Model

The model can be deployed as a strategic tool by the USPTO to anticipate and preemptively address examiner attrition. By identifying examiners with a high probability of turnover, targeted interventions can be planned to improve job satisfaction and retention rates

```
#install.packages(c("arrow","gender", "wru", "lubridate", "gtsummary"))
# Load required libraries
library(gender)
```

```
## Warning: package 'gender' was built under R version 4.2.3
```

```
library(wru)
```

```
## Warning: package 'wru' was built under R version 4.2.3
```

```
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 4.2.3

##
## Attaching package: 'lubridate'

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

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.2.3

##
## 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
```

```
library(gtsummary)
```

```
## Warning: package 'gtsummary' was built under R version 4.2.3
```

```
library(arrow)
```

```
## Warning: package 'arrow' was built under R version 4.2.3

##
## Attaching package: 'arrow'

## The following object is masked from 'package:lubridate':
##
##     duration

## The following object is masked from 'package:utils':
##
##     timestamp
```

```
library(tidyr)
```

```
## Warning: package 'tidyr' was built under R version 4.2.3
```

```
library(zoo)
```

```
## Warning: package 'zoo' was built under R version 4.2.3
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(purrr)
```

```
## Warning: package 'purrr' was built under R version 4.2.3
```

```
data<- read_feather("D:\\Google Drive\\McGill\\Winter Semester\\ORGB - 671\\ta-assignments\\2023-ta-ass
```

```
# Task 1: Create individual-level variables
examiner_names <- data %>% distinct(examiner_name_first)

examiner_names
```

```
## # A tibble: 2,595 x 1
##    examiner_name_first
##    <chr>
##  1 JACQUELINE
##  2 BEKIR
##  3 CYNTHIA
##  4 MARY
##  5 MICHAEL
##  6 LINDA
##  7 KARA
##  8 VANESSA
##  9 TERESA
## 10 SUN
## # i 2,585 more rows
```

```
# get a table of names and gender
examiner_names_gender <- examiner_names %>%
  do(results = gender(.$examiner_name_first, method = "ssa")) %>%
  unnest(cols = c(results), keep_empty = TRUE) %>%
  select(
    examiner_name_first = name,
    gender,
    proportion_female
  )

examiner_names_gender
```

```
## # A tibble: 1,822 x 3
##    examiner_name_first gender proportion_female
##    <chr>               <chr>              <dbl>
##  1 AARON               male              0.0082
##  2 ABDEL               male              0
##  3 ABDOU               male              0
##  4 ABDUL               male              0
##  5 ABDULHAKIM          male              0
##  6 ABDULLAH            male              0
##  7 ABDULLAHI           male              0
##  8 ABIGAIL             female            0.998
##  9 ABIMBOLA            female            0.944
## 10 ABRAHAM             male              0.0031
## # i 1,812 more rows
```

```
# remove extra colums from the gender table
examiner_names_gender <- examiner_names_gender %>%
  select(examiner_name_first, gender)

# joining gender back to the dataset
data <- data %>%
  left_join(examiner_names_gender, by = "examiner_name_first")

# cleaning up
rm(examiner_names)
rm(examiner_names_gender)
gc()
```

```
##            used  (Mb) gc trigger  (Mb)  max used  (Mb)
## Ncells  4469059 238.7    8130994 434.3   4490813 239.9
## Vcells 59441804 453.6  124296562 948.4 103887488 792.6
```

```
library(wru)

examiner_surnames <- data %>%
  select(surname = examiner_name_last) %>%
  distinct()

examiner_surnames
```

```
## # A tibble: 3,806 x 1
##    surname
##    <chr>
##  1 HOWARD
##  2 YILDIRIM
##  3 HAMILTON
##  4 MOSHER
##  5 BARR
##  6 GRAY
##  7 MCMILLIAN
##  8 FORD
##  9 STRZELECKA
## 10 KIM
## # i 3,796 more rows
```

```r
examiner_race <- predict_race(voter.file = examiner_surnames, surname.only = T) %>%
  as_tibble()
```

```
## Warning: Unknown or uninitialised column: 'state'.
```

```
## Proceeding with last name predictions...
```

```
## i All local files already up-to-date!
```

```
## 701 (18.4%) individuals' last names were not matched.
```

```r
examiner_race
```

```
## # A tibble: 3,806 x 6
##    surname    pred.whi pred.bla pred.his pred.asi pred.oth
##    <chr>         <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
##  1 HOWARD       0.597   0.295    0.0275   0.00690   0.0741
##  2 YILDIRIM     0.807   0.0273   0.0694   0.0165    0.0798
##  3 HAMILTON     0.656   0.239    0.0286   0.00750   0.0692
##  4 MOSHER       0.915   0.00425  0.0291   0.00917   0.0427
##  5 BARR         0.784   0.120    0.0268   0.00830   0.0615
##  6 GRAY         0.640   0.252    0.0281   0.00748   0.0724
##  7 MCMILLIAN    0.322   0.554    0.0212   0.00340   0.0995
##  8 FORD         0.576   0.320    0.0275   0.00621   0.0697
##  9 STRZELECKA   0.472   0.171    0.220    0.0825    0.0543
## 10 KIM          0.0169  0.00282  0.00546  0.943     0.0319
## # i 3,796 more rows
```

```r
examiner_race <- examiner_race %>%
  mutate(max_race_p = pmax(pred.asi, pred.bla, pred.his, pred.oth, pred.whi)) %>%
  mutate(race = case_when(
    max_race_p == pred.asi ~ "Asian",
    max_race_p == pred.bla ~ "black",
    max_race_p == pred.his ~ "Hispanic",
    max_race_p == pred.oth ~ "other",
    max_race_p == pred.whi ~ "white",
    TRUE ~ NA_character_
  ))

examiner_race
```

```
## # A tibble: 3,806 x 8
##    surname    pred.whi pred.bla pred.his pred.asi pred.oth max_race_p race
##    <chr>         <dbl>    <dbl>    <dbl>    <dbl>    <dbl>      <dbl> <chr>
##  1 HOWARD       0.597   0.295    0.0275   0.00690   0.0741      0.597 white
##  2 YILDIRIM     0.807   0.0273   0.0694   0.0165    0.0798      0.807 white
##  3 HAMILTON     0.656   0.239    0.0286   0.00750   0.0692      0.656 white
##  4 MOSHER       0.915   0.00425  0.0291   0.00917   0.0427      0.915 white
##  5 BARR         0.784   0.120    0.0268   0.00830   0.0615      0.784 white
##  6 GRAY         0.640   0.252    0.0281   0.00748   0.0724      0.640 white
##  7 MCMILLIAN    0.322   0.554    0.0212   0.00340   0.0995      0.554 black
```

```
##  8 FORD          0.576   0.320    0.0275   0.00621   0.0697      0.576 white
##  9 STRZELECKA    0.472   0.171    0.220    0.0825    0.0543      0.472 white
## 10 KIM           0.0169  0.00282  0.00546  0.943     0.0319      0.943 Asian
## # i 3,796 more rows
```

```
# removing extra columns
examiner_race <- examiner_race %>%
  select(surname,race)

data <- data %>%
  left_join(examiner_race, by = c("examiner_name_last" = "surname"))

rm(examiner_race)
rm(examiner_surnames)
gc()
```

```
##            used  (Mb) gc trigger  (Mb)  max used  (Mb)
## Ncells  4579216 244.6    8130994 434.3   6600586 352.6
## Vcells 61663541 470.5  124296562 948.4 123443408 941.8
```

```
library(lubridate) # to work with dates

examiner_dates <- data %>%
  select(examiner_id, filing_date, appl_status_date)

examiner_dates
```

```
## # A tibble: 2,018,477 x 3
##    examiner_id filing_date appl_status_date
##          <dbl> <date>      <chr>
##  1       96082 2000-01-26  30jan2003 00:00:00
##  2       87678 2000-10-11  27sep2010 00:00:00
##  3       63213 2000-05-17  30mar2009 00:00:00
##  4       73788 2001-07-20  07sep2009 00:00:00
##  5       77294 2000-04-10  19apr2001 00:00:00
##  6       68606 2000-04-28  16jul2001 00:00:00
##  7       89557 2004-01-26  15may2017 00:00:00
##  8       97543 2000-06-23  03apr2002 00:00:00
##  9       98714 2000-02-04  27nov2002 00:00:00
## 10       65530 2002-02-20  23mar2009 00:00:00
## # i 2,018,467 more rows
```

```
examiner_dates <- examiner_dates %>%
  mutate(start_date = ymd(filing_date), end_date = as_date(dmy_hms(appl_status_date)))
```

```
examiner_dates <- examiner_dates %>%
  group_by(examiner_id) %>%
  summarise(
    earliest_date = min(start_date, na.rm = TRUE),
    latest_date = max(end_date, na.rm = TRUE),
    tenure_days = interval(earliest_date, latest_date) %/% days(1)
    ) %>%
```

```
  filter(year(latest_date)<2018)

examiner_dates
```

```
## # A tibble: 5,625 x 4
##    examiner_id earliest_date latest_date tenure_days
##          <dbl> <date>        <date>            <dbl>
##  1       59012 2004-07-28    2015-07-24         4013
##  2       59025 2009-10-26    2017-05-18         2761
##  3       59030 2005-12-12    2017-05-22         4179
##  4       59040 2007-09-11    2017-05-23         3542
##  5       59052 2001-08-21    2007-02-28         2017
##  6       59054 2000-11-10    2016-12-23         5887
##  7       59055 2004-11-02    2007-12-26         1149
##  8       59056 2000-03-24    2017-05-22         6268
##  9       59074 2000-01-31    2017-03-17         6255
## 10       59081 2011-04-21    2017-05-19         2220
## # i 5,615 more rows
```

```
data <- data %>%
  left_join(examiner_dates, by = "examiner_id")

rm(examiner_dates)
gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  4585834 245.0   14736320  787.1  14736320  787.1
## Vcells 74025535 564.8  149235874 1138.6 149215654 1138.5
```

```
data <- data %>%
  select(
    application_number,
    filing_date,
    examiner_name_last,
    examiner_name_first,
    examiner_name_middle,
    examiner_id,
    examiner_art_unit,
    uspc_class,
    uspc_subclass,
    patent_number,
    patent_issue_date,
    abandon_date,
    disposal_type,
    appl_status_code,
    appl_status_date,
    tc,
    gender = gender.y,  # Renaming the column to remove the suffix
    race = race.y,      # Renaming the column to remove the suffix
    earliest_date = earliest_date.y,  # Renaming the column to remove the suffix
    latest_date = latest_date.y,  # Renaming the column to remove the suffix
    tenure_days = tenure_days.y   # Renaming the column to remove the suffix
  )
```

Task 2: Create a panel dataset

```r
library(dplyr)
library(lubridate)
library(zoo)

# Assuming 'data' is your dataframe
# Convert dates to quarters
data <- data %>%
  mutate(
    filing_year_quarter = as.yearqtr(filing_date),
    abandon_year_quarter = as.yearqtr(abandon_date),
    issue_year_quarter = as.yearqtr(patent_issue_date)
  )

# Aggregate applications data by quarter
panel_data <- data %>%
  group_by(examiner_id, filing_year_quarter) %>%
  summarise(
    num_new_applications = n_distinct(application_number),
    num_abandoned_applications = sum(disposal_type == "ABN", na.rm = TRUE),
    num_issued_patents = sum(disposal_type == "ISS", na.rm = TRUE),
    num_in_process_applications = sum(disposal_type == "PEND", na.rm = TRUE),
    current_art_unit = first(examiner_art_unit),
    .groups = 'drop'
  )

# Add the count of people and women in each art unit per quarter
art_unit_info <- data %>%
  group_by(filing_year_quarter, examiner_art_unit) %>%
  summarise(
    num_people_in_art_unit = n_distinct(examiner_id),
    num_women_in_art_unit = sum(gender == "female", na.rm = TRUE),
    .groups = 'drop'
  )

# Join the art unit info with the main panel data
panel_data <- panel_data %>%
  left_join(art_unit_info, by = c("filing_year_quarter", "current_art_unit" = "examiner_art_unit"))

# Mark the last five quarters for each examiner and create separation_indicator
panel_data <- panel_data %>%
  group_by(examiner_id) %>%
  mutate(
    # Get the maximum quarter date for each examiner
    max_quarter_date = max(filing_year_quarter),
    # Check if the current quarter date is less than the maximum quarter date
    separation_indicator = if_else(filing_year_quarter < max_quarter_date, 0, 1)
  ) %>%
  ungroup()
```

```r
# Detect changes in current_art_unit
panel_data <- panel_data %>%
  group_by(examiner_id) %>%
  mutate(
    AU_move_indicator = if_else(current_art_unit != lag(current_art_unit, default = NA), 1, 0),
    AU_move_indicator = replace_na(AU_move_indicator, 0)
  ) %>%
  ungroup()
```

```r
table(panel_data$separation_indicator)
```

```
##
##      0      1
## 185232   5649
```

Task 3: Prediction Model

```r
#install.packages('xgboost')
# Load necessary libraries
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 4.2.3
```

```
##
## Attaching package: 'xgboost'
```

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

```r
library(dplyr)
library(pROC)
```

```
## Warning: package 'pROC' was built under R version 4.2.3
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
# Prepare the data for regression
regression_data <- panel_data %>%
  filter(num_new_applications > 0) %>%
  select(separation_indicator, num_new_applications, num_abandoned_applications,
         num_issued_patents, num_people_in_art_unit, num_women_in_art_unit)

# Split data into training and testing sets
set.seed(123) # For reproducibility
train_index <- sample(1:nrow(regression_data), 0.8 * nrow(regression_data))
train_data <- regression_data[train_index, ]
test_data <- regression_data[-train_index, ]

# Convert training data to xgb.DMatrix format
train_matrix <- xgboost::xgb.DMatrix(data = as.matrix(train_data[, -which(names(train_data) == "separat
                                     label = train_data$separation_indicator)

# Parameters for XGBoost
params <- list(objective = "binary:logistic", eval_metric = "auc")

# 5-fold cross-validation on training data
cv_results <- xgboost::xgb.cv(params = params, data = train_matrix, nrounds = 100, nfold = 5, showsd =
```

```
## [1]  train-auc:0.929423+0.001611 test-auc:0.926861+0.005757
## [11] train-auc:0.944248+0.000837 test-auc:0.938181+0.002930
## [21] train-auc:0.947628+0.000731 test-auc:0.939215+0.002976
## [31] train-auc:0.948919+0.000680 test-auc:0.939618+0.002943
## [41] train-auc:0.951252+0.000879 test-auc:0.940126+0.002796
## [51] train-auc:0.953153+0.000735 test-auc:0.940219+0.002557
## [61] train-auc:0.955547+0.000555 test-auc:0.940120+0.002710
## [71] train-auc:0.956942+0.000883 test-auc:0.940257+0.002651
## [81] train-auc:0.958129+0.000750 test-auc:0.940074+0.002493
## [91] train-auc:0.959314+0.000526 test-auc:0.939973+0.002407
## [100]    train-auc:0.960298+0.000443 test-auc:0.939809+0.002282
```

```r
# Review the cross-validation results
cv_results
```

```
## ##### xgb.cv 5-folds
##  iter train_auc_mean train_auc_std test_auc_mean test_auc_std
##     1      0.9294226  0.0016109651     0.9268610  0.005757441
##     2      0.9312883  0.0016527971     0.9283892  0.005774529
##     3      0.9329197  0.0017440401     0.9297907  0.005995909
##     4      0.9380090  0.0019887163     0.9343959  0.004438283
##     5      0.9402174  0.0008805072     0.9355848  0.004840221
##     6      0.9416320  0.0010929389     0.9368707  0.003565305
##     7      0.9424175  0.0009023171     0.9374707  0.003082215
##     8      0.9427399  0.0010124501     0.9374840  0.002789797
##     9      0.9433618  0.0008273794     0.9378484  0.002856248
##    10      0.9437144  0.0008621785     0.9380531  0.002907550
##    11      0.9442478  0.0008372799     0.9381808  0.002929849
##    12      0.9448408  0.0007097422     0.9384413  0.003188950
##    13      0.9453808  0.0006661730     0.9387727  0.003293758
##    14      0.9456400  0.0006622338     0.9389262  0.003213537
```

```
##     15      0.9460183  0.0006982130     0.9391444  0.002825001
##     16      0.9463991  0.0007608093     0.9391095  0.002869206
##     17      0.9467012  0.0007024359     0.9391160  0.002873782
##     18      0.9469335  0.0007039782     0.9391922  0.002969855
##     19      0.9471336  0.0007637376     0.9391751  0.002976273
##     20      0.9474623  0.0007109273     0.9392173  0.002955655
##     21      0.9476284  0.0007305995     0.9392147  0.002976262
##     22      0.9477106  0.0007385817     0.9392868  0.002911290
##     23      0.9478241  0.0007188191     0.9393242  0.002905447
##     24      0.9479064  0.0006961560     0.9393070  0.002863340
##     25      0.9480303  0.0007130090     0.9393744  0.002828150
##     26      0.9481765  0.0007265137     0.9393890  0.002919109
##     27      0.9482360  0.0007251882     0.9393650  0.002919303
##     28      0.9483968  0.0007057960     0.9394198  0.002941787
##     29      0.9484933  0.0006948142     0.9394855  0.002941289
##     30      0.9486687  0.0006359755     0.9395595  0.002954598
##     31      0.9489186  0.0006801323     0.9396182  0.002942962
##     32      0.9490954  0.0007773320     0.9396853  0.002893647
##     33      0.9492388  0.0008268728     0.9396582  0.002856169
##     34      0.9494939  0.0007186917     0.9397537  0.002874200
##     35      0.9496394  0.0007348472     0.9397466  0.002831314
##     36      0.9499639  0.0007966036     0.9398434  0.002857362
##     37      0.9504465  0.0007207632     0.9399765  0.002871312
##     38      0.9506151  0.0008605489     0.9399982  0.002810920
##     39      0.9507920  0.0007434670     0.9399849  0.002779115
##     40      0.9509158  0.0007721132     0.9399739  0.002780654
##     41      0.9512519  0.0008791128     0.9401258  0.002795930
##     42      0.9514319  0.0007811434     0.9400704  0.002781435
##     43      0.9516622  0.0007889319     0.9400035  0.002759012
##     44      0.9518682  0.0008077051     0.9400505  0.002739914
##     45      0.9520617  0.0008335236     0.9401185  0.002734469
##     46      0.9521831  0.0008517604     0.9401308  0.002745582
##     47      0.9523553  0.0007447472     0.9401056  0.002758719
##     48      0.9525468  0.0007609964     0.9401430  0.002696489
##     49      0.9528664  0.0009336810     0.9402486  0.002602253
##     50      0.9530658  0.0007346794     0.9402480  0.002619390
##     51      0.9531531  0.0007349852     0.9402194  0.002557460
##     52      0.9534165  0.0007516511     0.9402500  0.002642029
##     53      0.9537116  0.0007079342     0.9402330  0.002678946
##     54      0.9540237  0.0008238101     0.9402021  0.002642859
##     55      0.9542168  0.0008378497     0.9401502  0.002609717
##     56      0.9545291  0.0008006400     0.9401563  0.002656091
##     57      0.9547350  0.0008505029     0.9401256  0.002660969
##     58      0.9549345  0.0007671885     0.9401135  0.002644713
##     59      0.9551534  0.0005773215     0.9401323  0.002696509
##     60      0.9553530  0.0006518800     0.9401366  0.002700552
##     61      0.9555470  0.0005553613     0.9401201  0.002710498
##     62      0.9557387  0.0005825695     0.9401696  0.002703506
##     63      0.9558684  0.0006129711     0.9402263  0.002702076
##     64      0.9561405  0.0006937740     0.9402330  0.002652747
##     65      0.9562472  0.0007396519     0.9402633  0.002638200
##     66      0.9564246  0.0007603991     0.9402774  0.002716877
##     67      0.9565764  0.0008317134     0.9403016  0.002719531
##     68      0.9567466  0.0009119584     0.9403195  0.002690157
```

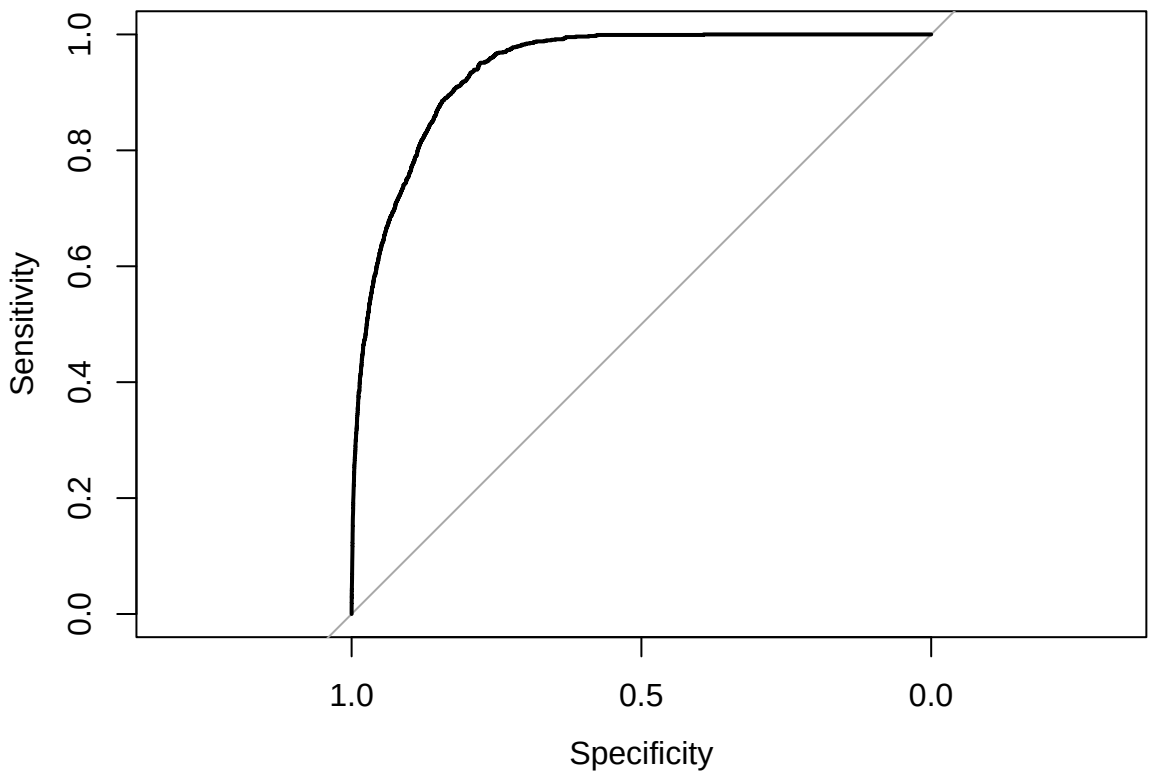
```
##    69      0.9567814  0.0009136925     0.9403126  0.002662627
##    70      0.9569086  0.0008706055     0.9402660  0.002636275
##    71      0.9569416  0.0008825354     0.9402570  0.002650655
##    72      0.9570230  0.0008839880     0.9402661  0.002656954
##    73      0.9572159  0.0007656031     0.9402597  0.002687845
##    74      0.9573435  0.0007324137     0.9402695  0.002683804
##    75      0.9574514  0.0006842496     0.9402419  0.002672624
##    76      0.9576018  0.0006546873     0.9401929  0.002654550
##    77      0.9577687  0.0007193676     0.9401554  0.002603945
##    78      0.9578565  0.0007720522     0.9401183  0.002620192
##    79      0.9579164  0.0007554263     0.9400822  0.002595213
##    80      0.9580257  0.0007332649     0.9400618  0.002509583
##    81      0.9581290  0.0007496759     0.9400743  0.002492799
##    82      0.9582195  0.0007525874     0.9400359  0.002507518
##    83      0.9583846  0.0007017311     0.9400426  0.002527570
##    84      0.9584822  0.0007174246     0.9400613  0.002506513
##    85      0.9586156  0.0006773619     0.9400355  0.002488677
##    86      0.9587406  0.0007091230     0.9400161  0.002479619
##    87      0.9589121  0.0006419124     0.9399891  0.002476801
##    88      0.9590709  0.0005769003     0.9399619  0.002467260
##    89      0.9591188  0.0005428059     0.9399534  0.002460598
##    90      0.9592418  0.0005562006     0.9399686  0.002420752
##    91      0.9593140  0.0005263737     0.9399729  0.002407067
##    92      0.9594335  0.0004658150     0.9399725  0.002412718
##    93      0.9595497  0.0003969520     0.9399966  0.002422468
##    94      0.9596994  0.0004347248     0.9399850  0.002457770
##    95      0.9597565  0.0004493097     0.9399451  0.002398925
##    96      0.9598494  0.0004760700     0.9399485  0.002352397
##    97      0.9599930  0.0005322298     0.9398942  0.002320210
##    98      0.9600783  0.0005645626     0.9398752  0.002298659
##    99      0.9602160  0.0005171617     0.9398379  0.002315556
##   100      0.9602976  0.0004434064     0.9398091  0.002282259
##  iter train_auc_mean train_auc_std test_auc_mean test_auc_std
```

```r
best_nrounds <- 100

# Train the final model on the full training set
final_model <- xgboost::xgb.train(params = params, data = train_matrix, nrounds = best_nrounds)

# Convert test data to xgb.DMatrix format
test_matrix <- xgboost::xgb.DMatrix(data = as.matrix(test_data[, -which(names(test_data) == "separation_
                                    label = test_data$separation_indicator)

# Predict on test data
test_predictions <- predict(final_model, test_matrix)

# Evaluate the model on the testing set
# Plot ROC curve and calculate AUC
roc_curve <- roc(test_data$separation_indicator, test_predictions)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
plot(roc_curve)
```




```r
auc_score <- auc(roc_curve)

# Print AUC
print(paste("AUC:", auc_score))
```

```
## [1] "AUC: 0.939148615896406"
```

General Performance: The model's performance is improving with more iterations as both the mean AUC on training and test data are increasing. This suggests that additional boosting rounds are effectively reducing errors made by previous trees.

Overfitting Check: The training and test AUC scores are close to each other, and the gap does not widen significantly with more iterations. This indicates that the model is not overfitting the training data; rather, it is generalizing well to the unseen validation data in each fold of the cross-validation.

Stability: The standard deviations for both training and test AUC are relatively low, which implies that the model's performance is stable across different subsets of the data. As the iterations increase, these standard deviations tend to get even smaller, suggesting increased stability.

Optimal Iterations: By the 19th iteration, the improvement in test AUC scores begins to plateau, indicating that subsequent iterations beyond this point are yielding diminishing returns in terms of model performance on the test data