edld_final_project_analysis

Denicia Espinosa Aragon

2023-11-29

import data

```r
options(digits = 10)
stackoverflow_df_og <- rio::import("data/stackoverflow_full.csv")

stackoverflow_df_short<- stackoverflow_df_og
#str(stackoverflow_df_og)

### reduce the amount of data so it can run on personal computer
# Set a seed for reproducibility
set.seed(42)

# Randomly select 1000 rows
#stackoverflow_df_short<- stackoverflow_df_og[sample(nrow(stackoverflow_df_og), 70000, replace = FALSE)


str(stackoverflow_df_og)
```

```
## 'data.frame':    73462 obs. of  15 variables:
##  $ V1            : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ Age           : chr  "<35" "<35" "<35" "<35" ...
##  $ Accessibility : chr  "No" "No" "No" "No" ...
##  $ EdLevel       : chr  "Master" "Undergraduate" "Master" "Undergraduate" ...
##  $ Employment    : int  1 1 1 1 0 1 1 1 1 1 ...
##  $ Gender        : chr  "Man" "Man" "Man" "Man" ...
##  $ MentalHealth  : chr  "No" "No" "No" "No" ...
##  $ MainBranch    : chr  "Dev" "Dev" "Dev" "Dev" ...
##  $ YearsCode     : int  7 12 15 9 40 9 26 14 39 20 ...
##  $ YearsCodePro  : int  4 5 6 6 30 2 18 5 21 16 ...
##  $ Country       : chr  "Sweden" "Spain" "Germany" "Canada" ...
##  $ PreviousSalary: num  51552 46482 77290 46135 160932 ...
##  $ HaveWorkedWith: chr  "C++;Python;Git;PostgreSQL" "Bash/Shell;HTML/CSS;JavaScript;Node.js;SQL;TypeS
##  $ ComputerSkills: int  4 12 7 13 2 5 17 4 3 6 ...
##  $ Employed      : int  0 1 0 0 0 0 1 0 0 0 ...
```

```r
#str(stackoverflow_df_short)
#table(stackoverflow_df_short$Country)
```

creating initial dummy variables for "HaveWorkedWith" because the of the complexity of the data

```
# need outcome variable to be categorical
#stackoverflow_df_short$Employed <- as.factor(stackoverflow_df_short$Employed)

 # creating dummy variables for HaveWorkedWith
 # stackoverflow_workedWith <- stackoverflow_df_short %>%
 #   select("V1", "HaveWorkedWith") %>%
 #   tidyr::separate_rows(HaveWorkedWith, sep = ";") %>%
 #   filter(HaveWorkedWith != "") %>%
 #   mutate(value = 1) %>%
 #   pivot_wider(
 #     names_from = HaveWorkedWith,
 #     values_from = value,
 #     values_fill = 0
 #   ) %>%
 #   mutate(across(-V1, as.integer))
```

```
# Assuming your original dataframe is named 'df'
# If not, replace 'df' with your actual dataframe name

# Define a function to segment countries into continents
segment_country <- function(country) {
  if (country %in% c('United States of America', 'Canada', 'Mexico')) {
    return('NorthAmerica')
  } else if (country %in% c('United Kingdom of Great Britain and Northern Ireland', 'France', 'Germany'
    return('Europe')
  } else if (country %in% c('Brazil', 'Argentina', 'Chile', 'Colombia', 'Peru', 'Venezuela, Bolivarian R
    return('South America')
  } else if (country %in% c('China', 'Japan', 'South Korea', 'Viet Nam', 'India', 'Sri Lanka', 'Pakistan
    return('Asia')
  } else if (country %in% c('Australia', 'New Zealand', 'Fiji', 'Papua New Guinea', 'Solomon Islands',
    return('Australia')
  } else {
    return('Others')
  }
}

# Apply the function to create a new column 'Continent'
stackoverflow_df_short$Continent <- sapply(stackoverflow_df_short$Country, segment_country)
```

```
 # # creating dummy variables for HaveWorkedWith
 # stackoverflow_country <- stackoverflow_df_short %>%
 #   select("V1", "Country") %>%
 #   tidyr::separate_rows(Country, sep = ";") %>%
 #   filter(Country != "") %>%
 #   mutate(value = 1) %>%
 #   pivot_wider(
 #     names_from = Country,
 #     values_from = value,
 #     values_fill = 0
```

```r
#   ) %>%
#   mutate(across(-V1, as.integer))
```

creating all other dummy variables becasue it is not working in blueprint

```r
# Use dummy_cols to create dummy variables
# stackoverflow_df_dummy <- dummy_cols(stackoverflow_df_short, select_columns = c("Accessibility",  "Ag

# place HaveWorkedWith dummy variables into the full dataset
#stackoverflow_df_wide <- merge(stackoverflow_df_dummy, stackoverflow_workedWith, by = "V1")
# stackoverflow_df_wide <- merge(stackoverflow_df_short, stackoverflow_workedWith, by = "V1")
#
# remove "HaveWorkedWith" since there are not dummy variables of it
stackoverflow_df_wide <- stackoverflow_df_short %>%
  select(!c("HaveWorkedWith", "Country", "Accessibility", "Employment"))


# # Replace spaces with underscores in variable names
#names(stackoverflow_df_wide) <- gsub(" ", "_", names(stackoverflow_df_wide))
```

investigate missingness

this data set was already preprocessed and cleaned, but to double-check we will investigate missingness

```r
require(finalfit)

ff_glimpse(stackoverflow_df_wide)$Continuous[,c('n','missing_percent')]
```

```
##                    n missing_percent
## V1             73462             0.0
## YearsCode      73462             0.0
## YearsCodePro   73462             0.0
## PreviousSalary 73462             0.0
## ComputerSkills 73462             0.0
## Employed       73462             0.0
```

```r
ff_glimpse(stackoverflow_df_wide)$Categorical[,c('n','missing_percent')]
```

```
##                  n missing_percent
## Age          73462             0.0
## EdLevel      73462             0.0
## Gender       73462             0.0
## MentalHealth 73462             0.0
## MainBranch   73462             0.0
## Continent    73462             0.0
```

did not find any missingess in the data set for continuous and categorical variables

Logistic Regression with No Penalty

Logistic Regression with No Penalty

blueprint

```r
#(stackoverflow_df_short)
```

```r
categorical <- names(stackoverflow_df_wide)[sapply(stackoverflow_df_wide, is.character)]


# Print the list of categorical variables
print(categorical)
```

```
## [1] "Age"          "EdLevel"      "Gender"       "MentalHealth" "MainBranch"
## [6] "Continent"
```

```r
# blueprint <- recipe(x     = stackoverflow_df_wide,
#                      vars  = colnames(stackoverflow_df_wide),
#                    #  vars  = c(id,outcome,categorical,numeric), # declare variables
#                      roles = c('id',rep('predictor',11),'outcome', rep('predictor',118))) %>%
#    step_dummy(all_of(categorical),one_hot=TRUE) %>%
#    step_num2factor(Employed,
#                    transform = function(x) x + 1,
#                    levels=c('No','Yes'))

blueprint <- recipe(x     = stackoverflow_df_wide,
                    vars  = colnames(stackoverflow_df_wide),
                  #  vars  = c(id,outcome,categorical,numeric), # declare variables
                     roles = c('id',rep('predictor',9),'outcome', 'predictor')) %>%
   step_dummy(all_of(categorical),one_hot=TRUE) %>%
   step_num2factor(Employed,
                  transform = function(x) x + 1,
                  levels=c('No','Yes'))



 blueprint
```

```
##

## -- Recipe ----------------------------------------------------------------------

##

## -- Inputs

## Number of variables by role

## outcome:    1
## predictor: 10
## id:         1
```

```
##
## -- Operations
## * Dummy variables from: all_of(categorical)
## * Factor variables from: Employed
```

```
summary(blueprint)
```

```
## # A tibble: 12 x 4
##    variable       type      role      source
##    <chr>          <list>    <chr>     <chr>
##  1 V1             <chr [2]> id        original
##  2 Age            <chr [3]> predictor original
##  3 EdLevel        <chr [3]> predictor original
##  4 Gender         <chr [3]> predictor original
##  5 MentalHealth   <chr [3]> predictor original
##  6 MainBranch     <chr [3]> predictor original
##  7 YearsCode      <chr [2]> predictor original
##  8 YearsCodePro   <chr [2]> predictor original
##  9 PreviousSalary <chr [2]> predictor original
## 10 ComputerSkills <chr [2]> predictor original
## 11 Employed       <chr [2]> outcome   original
## 12 Continent      <chr [3]> predictor original
```

Split Dataset with 80-20 split

```
# split data 80-20
set.seed(10312022)  # for reproducibility

# loc      <- sample(1:nrow(stackoverflow_df_wide), round(nrow(stackoverflow_df_wide) * 0.8))
# stackoverflow_tr  <- stackoverflow_df_wide[loc, ]
# stackoverflow_te  <- stackoverflow_df_wide[-loc, ]
#
# dim(stackoverflow_tr)
#
# dim(stackoverflow_te)



# needed to separate the data
index <- createDataPartition(stackoverflow_df_wide$Employed, p = 0.8, list = FALSE)

stackoverflow_tr <- stackoverflow_df_wide[index, ]
stackoverflow_te <- stackoverflow_df_wide[-index, ]

dim(stackoverflow_tr)
```

```
## [1] 58770    12
```

```r
dim(stackoverflow_te)
```

```
## [1] 14692    12
```

10-Fold Cross-Validation with random shuffle

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

# Create 10 folds with equal size

      folds = cut(seq(1,nrow(stackoverflow_tr)),breaks=10,labels=FALSE)

# Create the list for each fold

      my.indices <- vector('list',10)
      for(i in 1:10){
        my.indices[[i]] <- which(folds!=i)
      }

cv <- trainControl(method          = "cv",
                   index           = my.indices,
                   classProbs      = TRUE,      # predicted probabilities (logistic log function)
                   summaryFunction = mnLogLoss) # loss function
```

Train Model To Predict Scores Using Linear Regression Without Any Regularization.

grid without regularization

```r
grid_np <- expand.grid(alpha = 0, lambda = 0)
grid_np
```

```
##   alpha lambda
## 1     0      0
```

Train testing dataset with unregularized logistic regression

```r
caret_np <- caret::train(blueprint,
                          data      = stackoverflow_tr,
                          method    = "glm",
                          family    = 'binomial',     # classification/binary outcome
                          metric    = 'logLoss',
                        # tuneGrid = grid_np,
                          trControl = cv)
```

```
## New names:
## New names:
## * `Age_X.35` -> `Age_X.35...13`
## * `Age_X.35` -> `Age_X.35...14`

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## New names:
## New names:
## New names:
## New names:
## * `Age_X.35` -> `Age_X.35...13`
## * `Age_X.35` -> `Age_X.35...14`

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## New names:
## New names:
## New names:
## New names:
## * `Age_X.35` -> `Age_X.35...13`
## * `Age_X.35` -> `Age_X.35...14`

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## New names:
## New names:
## New names:
## New names:
## * `Age_X.35` -> `Age_X.35...13`
## * `Age_X.35` -> `Age_X.35...14`

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## New names:
## New names:
## New names:
## New names:
## * `Age_X.35` -> `Age_X.35...13`
## * `Age_X.35` -> `Age_X.35...14`

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## New names:
## New names:
## New names:
## New names:
## * `Age_X.35` -> `Age_X.35...13`
## * `Age_X.35` -> `Age_X.35...14`

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## New names:
## New names:
## New names:
## New names:
## * ‘Age_X.35‘ -> ‘Age_X.35...13‘
## * ‘Age_X.35‘ -> ‘Age_X.35...14‘

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## New names:
## New names:
## New names:
## New names:
## * ‘Age_X.35‘ -> ‘Age_X.35...13‘
## * ‘Age_X.35‘ -> ‘Age_X.35...14‘

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## New names:
## New names:
## New names:
## New names:
## * ‘Age_X.35‘ -> ‘Age_X.35...13‘
## * ‘Age_X.35‘ -> ‘Age_X.35...14‘

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## New names:
## New names:
## New names:
## New names:
## * ‘Age_X.35‘ -> ‘Age_X.35...13‘
## * ‘Age_X.35‘ -> ‘Age_X.35...14‘

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## New names:
## New names:
## New names:
## New names:
## * ‘Age_X.35‘ -> ‘Age_X.35...13‘
## * ‘Age_X.35‘ -> ‘Age_X.35...14‘

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
caret_np
```

```
## Generalized Linear Model
##
## 58770 samples
##    11 predictor
```

```
##     2 classes: 'No', 'Yes'
##
## Recipe steps: dummy, num2factor
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 52893, 52893, 52893, 52893, 52893, 52893, ...
## Resampling results:
##
##   logLoss
##   0.4437003472
```

Checking No Penalty Model Performance on Test Data

```r
options(digits = 10)
predicted_te_np <- predict(caret_np, newdata =stackoverflow_te, type='prob')
```

```
## New names:
## * `Age_X.35` -> `Age_X.35...13`
## * `Age_X.35` -> `Age_X.35...14`
```

```r
head(predicted_te_np)
```

```
##               No            Yes
## 1 0.44905219478 0.550947805222
## 2 0.99169173666 0.008308263336
## 3 0.14876872172 0.851231278277
## 4 0.96660605174 0.033393948264
## 5 0.38663562833 0.613364371671
## 6 0.01021203583 0.989787964169
```

```r
dim(predicted_te_np)
```

```
## [1] 14692     2
```

```r
head(predicted_te_np)
```

```
##               No            Yes
## 1 0.44905219478 0.550947805222
## 2 0.99169173666 0.008308263336
## 3 0.14876872172 0.851231278277
## 4 0.96660605174 0.033393948264
## 5 0.38663562833 0.613364371671
## 6 0.01021203583 0.989787964169
```

Evaluate and Report the Performance of the Unregularized Model on Test Dataset

LogLoss, AUC, Accuracy, True Negative Rate, False Yes Rate, True Yes Rate, Precision

```r
# Compute the AUC

cut.obj <- cutpointr(x     = predicted_te_np$Yes, # variable is coming from your predictions, here, it
                     class = stackoverflow_te$Employed,
                     na.rm = TRUE)
```

```
## Assuming the positive class is 1

## Assuming the positive class has higher x values
```

```r
auc_np <-auc(cut.obj)


# Confusion matrix assuming the threshold is 0.5

pred_class_np <- ifelse(predicted_te_np$Yes>.5,1,0)

confusion_np <- table(stackoverflow_te$Employed,pred_class_np)
confusion_np
```

```
##    pred_class_np
##        0    1
##   0 5237 1568
##   1 1611 6276
```

```r
# LogLoss
ll_np <- min(caret_np$results$logLoss)

# Accuracy (TP+TN/TP+TN+FP+FN)
# TP = confusion[2,2]
# TN = confusion[1,1]
# FP = confusion[1,2]
# FN = confusion[2,1]

acc_np <- (confusion_np[2,2] + confusion_np[1,1])/(confusion_np[2,2] + confusion_np[1,1]+ confusion_np[

# True Negative Rate (TN/TN+FP)

tnr_np <- confusion_np[1,1]/(confusion_np[1,1]+confusion_np[1,2])

# False Yes Rate

fpr_np <- confusion_np[1,2]/(confusion_np[1,1]+confusion_np[1,2])

# True Yes Rate (TP/TP+FN)

tpr_np <- confusion_np[2,2]/(confusion_np[2,1]+confusion_np[2,2])

# Precision

pre_np <- confusion_np[2,2]/(confusion_np[1,2]+confusion_np[2,2])
```

```r
# Create a data frame to store the results
results_np_df <- data.frame(
  Model = c("Non-Regularized Logistic Regression"),
  LL = c(ll_np),
  AUC = c(auc_np),
  ACC = c(acc_np),
  TPR = c(tpr_np),
  TNR = c(tnr_np),
  PRE = c(pre_np)
)

# Print the results data frame
print(results_np_df)
```

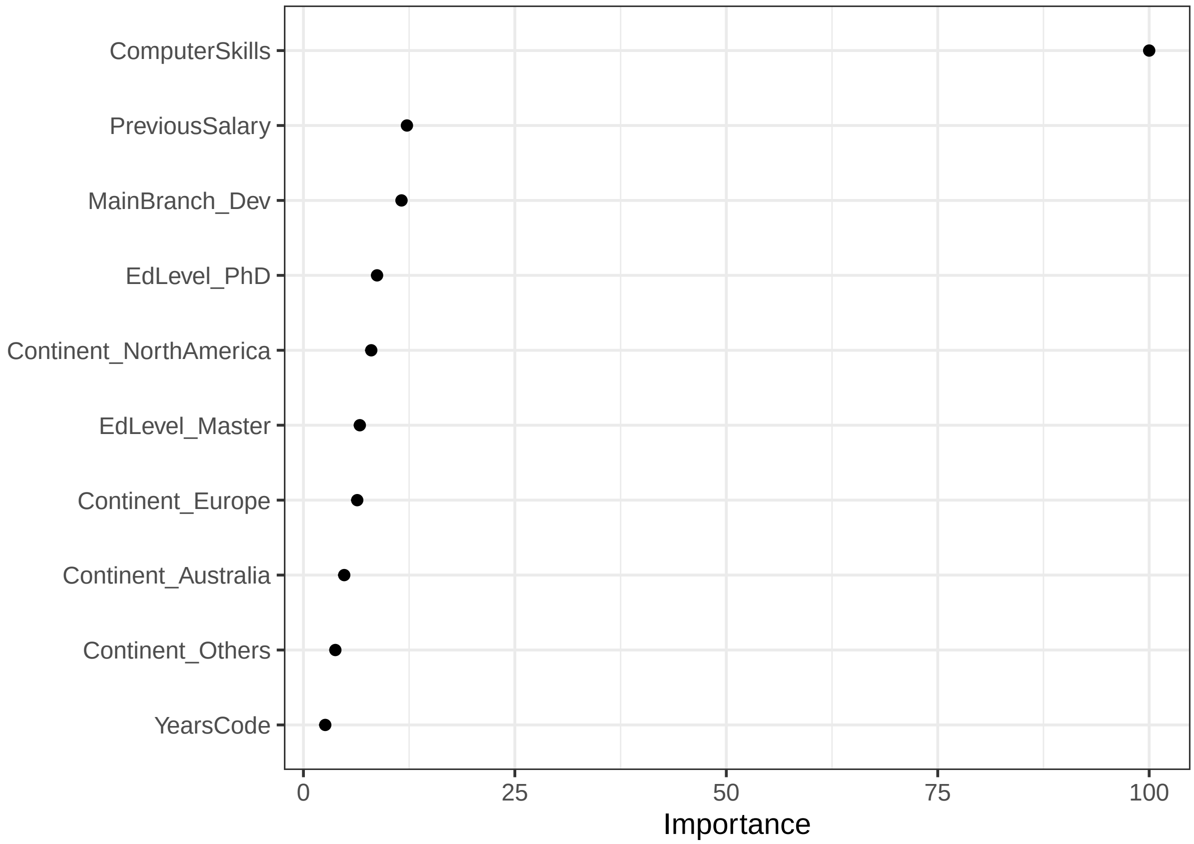
```
##                                 Model           LL          AUC          ACC
## 1 Non-Regularized Logistic Regression 0.4437003472 0.8713422426 0.7836237408
##           TPR          TNR          PRE
## 1 0.795739825 0.7695811903 0.8001019888
```

```r
library(vip)
```

```
##
## Attaching package: 'vip'
```

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

```r
vip_np <- vip(caret_np
    , num_features = 10, geom = "point") + theme_bw()
vip_np
```




```
saveRDS(vip_np, file = "/Users/daragon/Dropbox (University of Oregon)/courses/Fall 2023/EDLD653-ML/fina
```

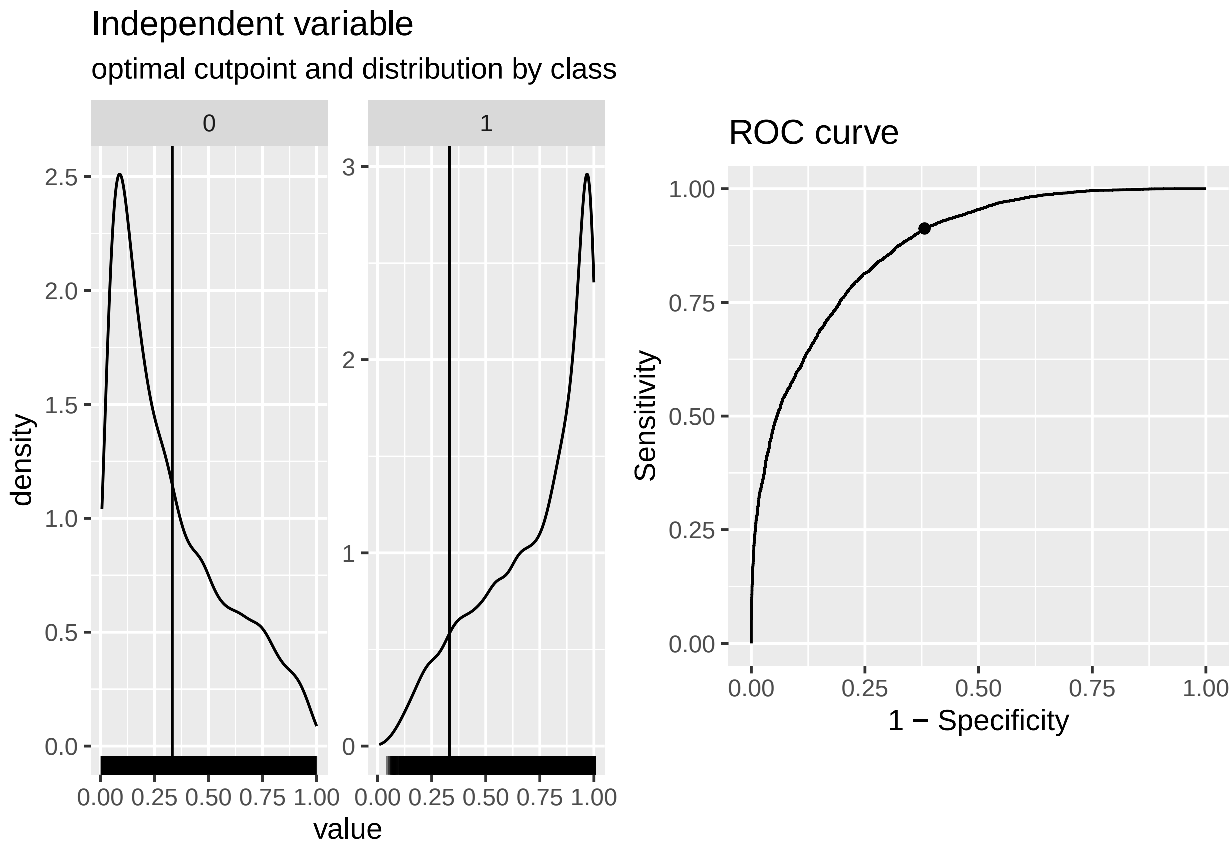
Finding an optimal cut-off value that maximizes a pre-defined metric

```
# Finding an optimal cut-off value that maximizes a pre-defined metric
cut.obj_np <- cutpointr(x     = predicted_te_np$Yes,
                        class = stackoverflow_te$Employed,
                        method = maximize_metric,
                        metric = F1_score,
                        na.rm = T)
```

```
## Assuming the positive class is 1
```

```
## Assuming the positive class has higher x values
```

```
#plot
plot(cut.obj_np)
```




```r
cut.obj_np$optimal_cutpoint
```

```
## [1] 0.3321998986
```

Logistic Regression with Ridge Penalty

convert all new HaveWorkedWith variables to factors for unregularized analysis

glm needs them as factors. glmnet needs them as integers

Logistic Regression with Ridge Penalty

grid with ridge regression

```r
# from 0.01 to 3 with increments of 0.01.
grid_ridge <- data.frame(alpha = 0, lambda = c(seq(0,.001,.00001),.005,.01,.05,.1))
grid_ridge
```

```
##     alpha  lambda
## 1       0 0.00000
## 2       0 0.00001
## 3       0 0.00002
```

```
## 4         0 0.00003
## 5         0 0.00004
## 6         0 0.00005
## 7         0 0.00006
## 8         0 0.00007
## 9         0 0.00008
## 10        0 0.00009
## 11        0 0.00010
## 12        0 0.00011
## 13        0 0.00012
## 14        0 0.00013
## 15        0 0.00014
## 16        0 0.00015
## 17        0 0.00016
## 18        0 0.00017
## 19        0 0.00018
## 20        0 0.00019
## 21        0 0.00020
## 22        0 0.00021
## 23        0 0.00022
## 24        0 0.00023
## 25        0 0.00024
## 26        0 0.00025
## 27        0 0.00026
## 28        0 0.00027
## 29        0 0.00028
## 30        0 0.00029
## 31        0 0.00030
## 32        0 0.00031
## 33        0 0.00032
## 34        0 0.00033
## 35        0 0.00034
## 36        0 0.00035
## 37        0 0.00036
## 38        0 0.00037
## 39        0 0.00038
## 40        0 0.00039
## 41        0 0.00040
## 42        0 0.00041
## 43        0 0.00042
## 44        0 0.00043
## 45        0 0.00044
## 46        0 0.00045
## 47        0 0.00046
## 48        0 0.00047
## 49        0 0.00048
## 50        0 0.00049
## 51        0 0.00050
## 52        0 0.00051
## 53        0 0.00052
## 54        0 0.00053
## 55        0 0.00054
## 56        0 0.00055
## 57        0 0.00056
```

```
## 58        0 0.00057
## 59        0 0.00058
## 60        0 0.00059
## 61        0 0.00060
## 62        0 0.00061
## 63        0 0.00062
## 64        0 0.00063
## 65        0 0.00064
## 66        0 0.00065
## 67        0 0.00066
## 68        0 0.00067
## 69        0 0.00068
## 70        0 0.00069
## 71        0 0.00070
## 72        0 0.00071
## 73        0 0.00072
## 74        0 0.00073
## 75        0 0.00074
## 76        0 0.00075
## 77        0 0.00076
## 78        0 0.00077
## 79        0 0.00078
## 80        0 0.00079
## 81        0 0.00080
## 82        0 0.00081
## 83        0 0.00082
## 84        0 0.00083
## 85        0 0.00084
## 86        0 0.00085
## 87        0 0.00086
## 88        0 0.00087
## 89        0 0.00088
## 90        0 0.00089
## 91        0 0.00090
## 92        0 0.00091
## 93        0 0.00092
## 94        0 0.00093
## 95        0 0.00094
## 96        0 0.00095
## 97        0 0.00096
## 98        0 0.00097
## 99        0 0.00098
## 100       0 0.00099
## 101       0 0.00100
## 102       0 0.00500
## 103       0 0.01000
## 104       0 0.05000
## 105       0 0.10000
```

Train testing dataset with unregularized logistic regression

```r
ridge <- caret::train(blueprint,
                           data      = stackoverflow_tr,
                           method    = "glmnet",
                           family    = 'binomial',      # classification/binary outcome
                           metric    = 'logLoss',
                          trControl = cv,
                          tuneGrid = grid_ridge)
```

```
## New names:
## Loading required package: Matrix
## Attaching package: 'Matrix'
## The following objects are masked from 'package:tidyr':
##
## expand, pack, unpack
## Loaded glmnet 4.1-8
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
```

```
## New names:
## New names:
## New names:
## * ‘Age_X.35‘ -> ‘Age_X.35...13‘
## * ‘Age_X.35‘ -> ‘Age_X.35...14‘
```

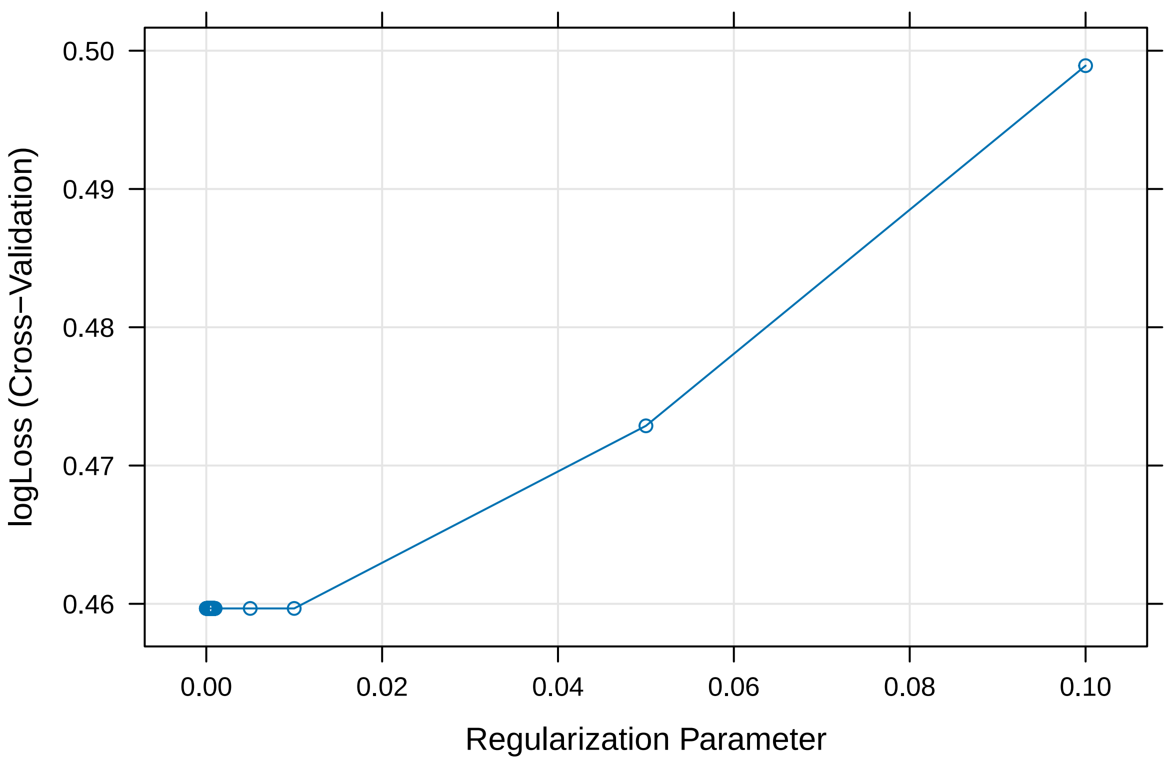
```
ridge
```

```
## glmnet
##
## 58770 samples
##    11 predictor
##     2 classes: 'No', 'Yes'
##
## Recipe steps: dummy, num2factor
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 52893, 52893, 52893, 52893, 52893, 52893, ...
## Resampling results across tuning parameters:
##
##   lambda   logLoss
##   0.00000  0.4596662447
##   0.00001  0.4596662447
##   0.00002  0.4596662447
##   0.00003  0.4596662447
##   0.00004  0.4596662447
##   0.00005  0.4596662447
##   0.00006  0.4596662447
##   0.00007  0.4596662447
##   0.00008  0.4596662447
##   0.00009  0.4596662447
##   0.00010  0.4596662447
##   0.00011  0.4596662447
##   0.00012  0.4596662447
##   0.00013  0.4596662447
##   0.00014  0.4596662447
##   0.00015  0.4596662447
##   0.00016  0.4596662447
##   0.00017  0.4596662447
##   0.00018  0.4596662447
##   0.00019  0.4596662447
##   0.00020  0.4596662447
##   0.00021  0.4596662447
##   0.00022  0.4596662447
##   0.00023  0.4596662447
##   0.00024  0.4596662447
##   0.00025  0.4596662447
##   0.00026  0.4596662447
##   0.00027  0.4596662447
##   0.00028  0.4596662447
##   0.00029  0.4596662447
##   0.00030  0.4596662447
##   0.00031  0.4596662447
##   0.00032  0.4596662447
##   0.00033  0.4596662447
```

```
##   0.00034  0.4596662447
##   0.00035  0.4596662447
##   0.00036  0.4596662447
##   0.00037  0.4596662447
##   0.00038  0.4596662447
##   0.00039  0.4596662447
##   0.00040  0.4596662447
##   0.00041  0.4596662447
##   0.00042  0.4596662447
##   0.00043  0.4596662447
##   0.00044  0.4596662447
##   0.00045  0.4596662447
##   0.00046  0.4596662447
##   0.00047  0.4596662447
##   0.00048  0.4596662447
##   0.00049  0.4596662447
##   0.00050  0.4596662447
##   0.00051  0.4596662447
##   0.00052  0.4596662447
##   0.00053  0.4596662447
##   0.00054  0.4596662447
##   0.00055  0.4596662447
##   0.00056  0.4596662447
##   0.00057  0.4596662447
##   0.00058  0.4596662447
##   0.00059  0.4596662447
##   0.00060  0.4596662447
##   0.00061  0.4596662447
##   0.00062  0.4596662447
##   0.00063  0.4596662447
##   0.00064  0.4596662447
##   0.00065  0.4596662447
##   0.00066  0.4596662447
##   0.00067  0.4596662447
##   0.00068  0.4596662447
##   0.00069  0.4596662447
##   0.00070  0.4596662447
##   0.00071  0.4596662447
##   0.00072  0.4596662447
##   0.00073  0.4596662447
##   0.00074  0.4596662447
##   0.00075  0.4596662447
##   0.00076  0.4596662447
##   0.00077  0.4596662447
##   0.00078  0.4596662447
##   0.00079  0.4596662447
##   0.00080  0.4596662447
##   0.00081  0.4596662447
##   0.00082  0.4596662447
##   0.00083  0.4596662447
##   0.00084  0.4596662447
##   0.00085  0.4596662447
##   0.00086  0.4596662447
##   0.00087  0.4596662447
```

```
##    0.00088  0.4596662447
##    0.00089  0.4596662447
##    0.00090  0.4596662447
##    0.00091  0.4596662447
##    0.00092  0.4596662447
##    0.00093  0.4596662447
##    0.00094  0.4596662447
##    0.00095  0.4596662447
##    0.00096  0.4596662447
##    0.00097  0.4596662447
##    0.00098  0.4596662447
##    0.00099  0.4596662447
##    0.00100  0.4596662447
##    0.00500  0.4596662447
##    0.01000  0.4596662447
##    0.05000  0.4728720251
##    0.10000  0.4989200759
##
## Tuning parameter 'alpha' was held constant at a value of 0
## logLoss was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 0 and lambda = 0.01.
```

```r
plot(ridge)
```



Checking No Penalty Model Performance on Test Data

```
predicted_te_ridge <- predict(ridge, newdata =stackoverflow_te, type='prob')
```

```
## New names:
## * `Age_X.35` -> `Age_X.35...13`
## * `Age_X.35` -> `Age_X.35...14`
```

```
head(predicted_te_ridge)
```

```
##               No           Yes
## 1 0.46096360819 0.53903639181
## 2 0.96686433375 0.03313566625
## 3 0.24672414494 0.75327585506
## 4 0.91495512742 0.08504487258
## 5 0.41762831946 0.58237168054
## 6 0.04531357792 0.95468642208
```

```
# Count NAs in the 'Yes' column of predicted_te_np data frame
na_count <- sum(is.na(predicted_te_ridge$Yes))

# Print the count
cat("Number of NAs in 'Yes' column:", na_count, "\n")
```

```
## Number of NAs in 'Yes' column: 0
```

```
dim(predicted_te_ridge)
```

```
## [1] 14692     2
```

```
head(predicted_te_ridge)
```

```
##               No           Yes
## 1 0.46096360819 0.53903639181
## 2 0.96686433375 0.03313566625
## 3 0.24672414494 0.75327585506
## 4 0.91495512742 0.08504487258
## 5 0.41762831946 0.58237168054
## 6 0.04531357792 0.95468642208
```

Evaluate and Report the Performance of the Unregularized Model on Test Dataset

LogLoss, AUC, Accuracy, True Negative Rate, False Yes Rate, True Yes Rate, Precision

```
# Compute the AUC

cut.obj <- cutpointr(x     = predicted_te_ridge$Yes, # variable is coming from your predictions, here,
                     class = stackoverflow_te$Employed,
                     na.rm = TRUE)
```

```
## Assuming the positive class is 1

## Assuming the positive class has higher x values
```

```
auc_ridge <-auc(cut.obj)


# Confusion matrix assuming the threshold is 0.5

pred_class_ridge <- ifelse(predicted_te_ridge$Yes>.5,1,0)

confusion_ridge <- table(stackoverflow_te$Employed,pred_class_ridge)
confusion_ridge
```

```
##    pred_class_ridge
##        0    1
##   0 5224 1581
##   1 1582 6305
```

```
# LogLoss
ll_ridge <- min(ridge$results$logLoss)

# Accuracy (TP+TN/TP+TN+FP+FN)
# TP = confusion[2,2]
# TN = confusion[1,1]
# FP = confusion[1,2]
# FN = confusion[2,1]

acc_ridge <- (confusion_ridge[2,2] + confusion_ridge[1,1])/(confusion_ridge[2,2] + confusion_ridge[1,1]+

# True Negative Rate (TN/TN+FP)

tnr_ridge <- confusion_ridge[1,1]/(confusion_ridge[1,1]+confusion_ridge[1,2])

# False Yes Rate

fpr_ridge <- confusion_ridge[1,2]/(confusion_ridge[1,1]+confusion_ridge[1,2])

# True Yes Rate (TP/TP+FN)

tpr_ridge <- confusion_ridge[2,2]/(confusion_ridge[2,1]+confusion_ridge[2,2])

# Precision

pre_ridge <- confusion_ridge[2,2]/(confusion_ridge[1,2]+confusion_ridge[2,2])


# Create a data frame to store the results
results_ridge_df <- data.frame(
  Model = c("Logistic Regression with Ridge Penalty"),
  LL = c(ll_ridge),
  AUC = c(auc_ridge),
  ACC = c(acc_ridge),
```

```r
  TPR = c(tpr_ridge),
  TNR = c(tnr_ridge),
  PRE = c(pre_ridge)
)

# Print the results data frame
print(results_ridge_df)
```

```
##                                   Model           LL          AUC          ACC
## 1 Logistic Regression with Ridge Penalty 0.4596662447 0.8706131715 0.7847127689
##            TPR          TNR          PRE
## 1 0.7994167618 0.7676708303 0.7995181334
```

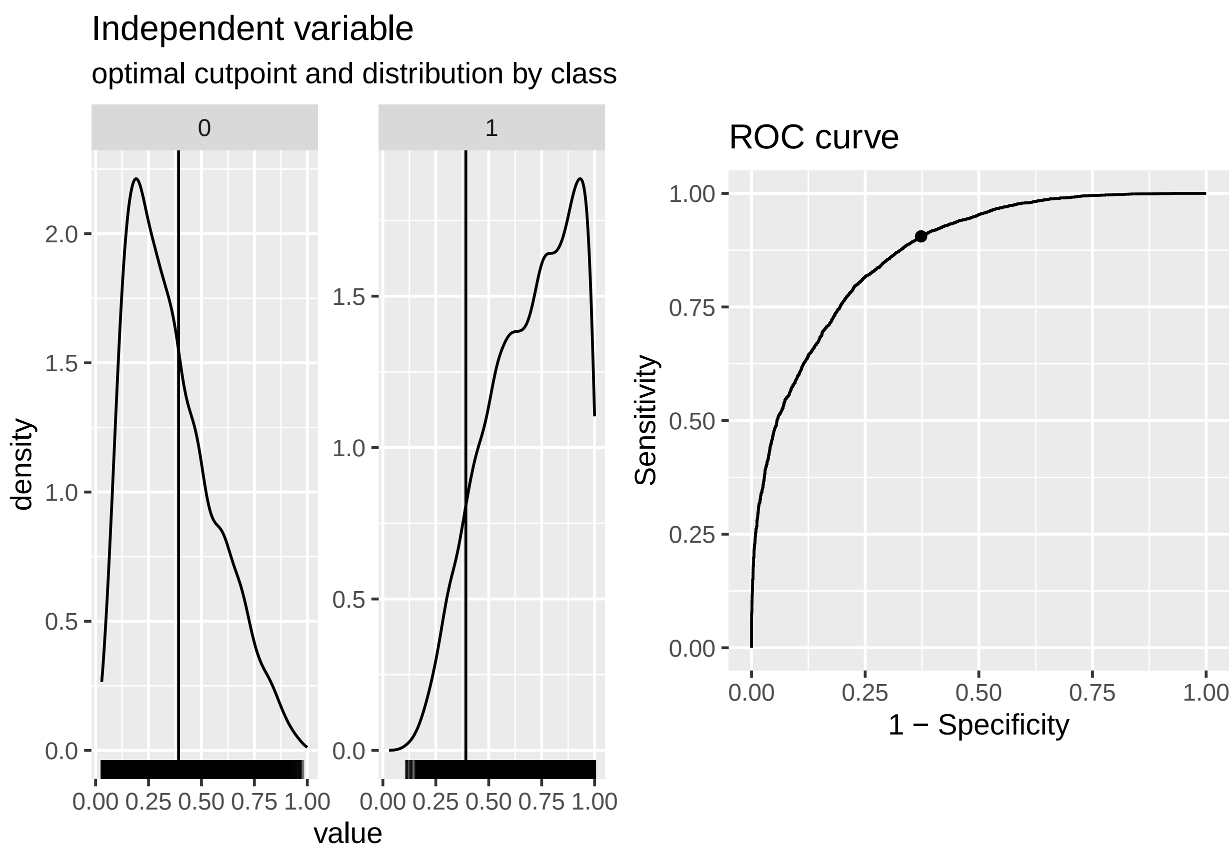
Finding an optimal cut-off value that maximizes a pre-defined metric

```r
# Finding an optimal cut-off value that maximizes a pre-defined metric
cut.obj_ridge <- cutpointr(x     = predicted_te_ridge$Yes,
                     class = stackoverflow_te$Employed,
                     method = maximize_metric,
                     metric = F1_score,
                     na.rm = T)
```

```
## Assuming the positive class is 1

## Assuming the positive class has higher x values
```

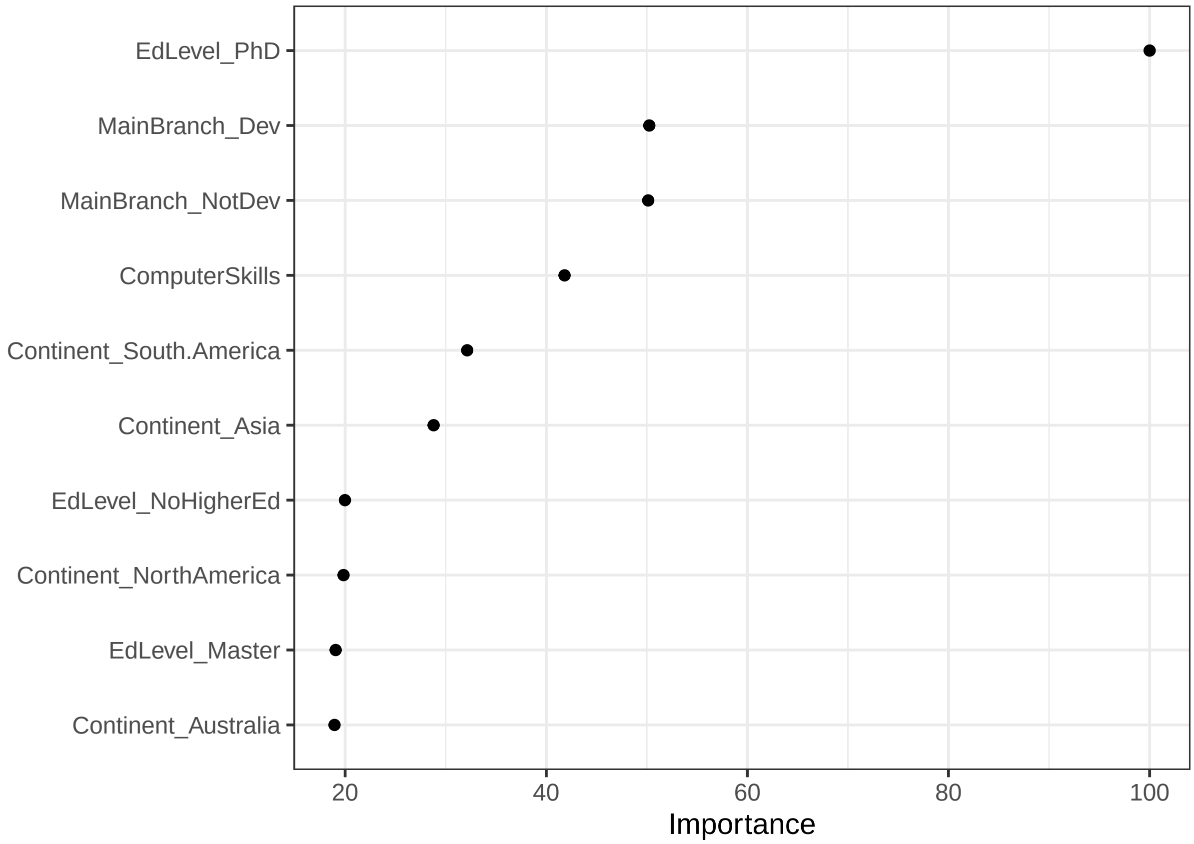
```r
#plot
plot(cut.obj_ridge)
```




```
cut.obj_ridge$optimal_cutpoint
```

```
## [1] 0.3920112453
```

```
library(vip)
vip_ridge <-  vip(ridge, num_features = 10, geom = "point") + theme_bw()
saveRDS(vip_ridge, file = "/Users/daragon/Dropbox (University of Oregon)/courses/Fall 2023/EDLD653-ML/f
vip_ridge
```



Bagged Decision Tree

```
library(rpart)
# Cross validation settings

    set.seed(10302021) # for reproducibility

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

  # Create 10 folds with equal size

    folds = cut(seq(1,nrow(stackoverflow_tr)),breaks=10,labels=FALSE)

  # Create the list for each fold

    my.indices <- vector('list',10)
    for(i in 1:10){
      my.indices[[i]] <- which(folds!=i)
    }


  cv <- trainControl(method = "cv",
                     index   = my.indices,
                     classProbs = TRUE,
                     summaryFunction = mnLogLoss)
```

```r
# Grid settings

  # Notice that I use **'gini'** for splitrule because this is
  # now a classification problem.

  grid <- expand.grid(mtry = 13,
                      splitrule='gini',
                      min.node.size=2)
  grid
```

```
##   mtry splitrule min.node.size
## 1   13      gini             2
```

```r
# Run the BAGGED Trees with different number of trees
# 5, 20, 40, 60, ..., 200

    nbags <- c(5,seq(20,200,20))

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

    for(i in 1:length(nbags)){

      bags[[i]] <- caret::train(blueprint,
                                data      = stackoverflow_tr,
                                method    = 'ranger',
                                trControl = cv,
                                tuneGrid  = grid,
                                metric    = 'logLoss',
                                num.trees = nbags[i],
                                max.depth = 60)
    }
```

```
## Loading required namespace: e1071

## Loading required namespace: ranger

## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
```

```
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
```

```
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
```

```
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
```

```
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
```

```
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
```

```
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
```

```
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
```

```
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
```

```
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## New names:
## * `Age_X.35` -> `Age_X.35...13`
## * `Age_X.35` -> `Age_X.35...14`
```

```
logLoss_ <- c()

for(i in 1:length(nbags)){

  logLoss_[i] = bags[[i]]$results$logLoss
}

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

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

```
# Predict the probabilities for the observations in the test dataset

predicted_te_bagged <- predict(bags[[11]], stackoverflow_te, type='prob')
```

```
## New names:
## * `Age_X.35` -> `Age_X.35...13`
## * `Age_X.35` -> `Age_X.35...14`
```

```
head(predicted_te_bagged)
```

```
##              No          Yes
## 1 0.3928235931 0.6071764069
## 2 0.9950000000 0.0050000000
## 3 0.0525000000 0.9475000000
## 4 1.0000000000 0.0000000000
## 5 0.3952576313 0.6047423687
## 6 0.0000000000 1.0000000000
```

Evaluate and Report the Performance of the Bagged Trees on Test Dataset

LogLoss, AUC, Accuracy, True Negative Rate, False Yes Rate, True Yes Rate, Precision

```r
# Compute the AUC

cut.obj_bagged <- cutpointr(x     = predicted_te_bagged$Yes, # variable is coming from your predictions
                        class = stackoverflow_te$Employed,
                        na.rm = TRUE)
```

```
## Assuming the positive class is 1

## Assuming the positive class has higher x values
```

```r
auc_bagged <-auc(cut.obj_bagged)


# Confusion matrix assuming the threshold is 0.5

pred_class_bagged <- ifelse(predicted_te_bagged$Yes>.5,1,0)

confusion_bagged <- table(stackoverflow_te$Employed,pred_class_bagged)
confusion_bagged
```

```
##    pred_class_bagged
##        0    1
##   0 4957 1848
##   1 1552 6335
```

```r
# Accuracy (TP+TN/TP+TN+FP+FN)
# TP = confusion[2,2]
# TN = confusion[1,1]
# FP = confusion[1,2]
# FN = confusion[2,1]

acc_bagged <- (confusion_bagged[2,2] + confusion_bagged[1,1])/(confusion_bagged[2,2] + confusion_bagged

# True Negative Rate (TN/TN+FP)

tnr_bagged <- confusion_bagged[1,1]/(confusion_bagged[1,1]+confusion_bagged[1,2])

# False Yes Rate

fpr_bagged <- confusion_bagged[1,2]/(confusion_bagged[1,1]+confusion_bagged[1,2])

# True Yes Rate (TP/TP+FN)

tpr_bagged <- confusion_bagged[2,2]/(confusion_bagged[2,1]+confusion_bagged[2,2])

# Precision

pre_bagged <- confusion_bagged[2,2]/(confusion_bagged[1,2]+confusion_bagged[2,2])


# Create a data frame to store the results
```

```r
results_bagged_df <- data.frame(
  Model = c("Logistic Regression with Bagged Trees"),
 # LL = c(ll_bagged),
  AUC = c(auc_bagged),
  ACC = c(acc_bagged),
  TPR = c(tpr_bagged),
  TNR = c(tnr_bagged),
  PRE = c(pre_bagged)
)

# Print the results data frame
print(results_bagged_df)
```

```
##                                   Model          AUC         ACC          TPR
## 1 Logistic Regression with Bagged Trees 0.8548394399 0.768581541 0.8032204894
##           TNR          PRE
## 1 0.7284349743 0.7741659538
```

```r


# Create a data frame to store the results
# Create a data frame to store the results
results_df <- data.frame(
  Model = c("Non-Regularized Logistic Regression", "Logistic Regression with Ridge Penalty", "Logistic R
  LL = c(ll_np, ll_ridge, ll_ridge),
  AUC = c(auc_np, auc_ridge, auc_bagged),
  ACC = c(acc_np, acc_ridge, acc_bagged),
  TPR = c(tpr_np, tpr_ridge, tpr_bagged),
  TNR = c(tnr_np, tnr_ridge, tnr_bagged),
  PRE = c(pre_np, pre_ridge, pre_bagged)
)

# Print the results data frame
print(results_df)
```

Full Table

```
##                                    Model           LL          AUC          ACC
## 1    Non-Regularized Logistic Regression 0.4437003472 0.8713422426 0.7836237408
## 2 Logistic Regression with Ridge Penalty 0.4596662447 0.8706131715 0.7847127689
## 3  Logistic Regression with Bagged Trees 0.4596662447 0.8548394399 0.7685815410
##           TPR          TNR          PRE
## 1 0.7957398250 0.7695811903 0.8001019888
## 2 0.7994167618 0.7676708303 0.7995181334
## 3 0.8032204894 0.7284349743 0.7741659538
```

```r
# results_df <- kbl(results_df, caption = "Table 2. Model Performace for All Models", booktabs = T) %>%
#   kable_styling(full_width = F) %>%
# column_spec(1, bold = F) %>%
# column_spec(2, width = "30em")
```

```r
saveRDS(results_df, file = "/Users/daragon/Dropbox (University of Oregon)/courses/Fall 2023/EDLD653-ML/


# Assuming results_df is a data frame you want to write to a file
# write.table(results_df, "/Users/daragon/Dropbox (University of Oregon)/courses/Fall 2023/EDLD653-ML/f
```