Predicting Wikipedia Vandalism Using CART & Random Forest

Kunpei Peng

2/6/2021

Step 1: Exploratory Analysis

First, we can see that 1815 vandalism cases were detected in the history of this page.

```
#Load data and name it Wiki
Wiki = read.csv("Wikipedia.csv")
str(Wiki)
```

```
## 'data.frame':    3876 obs. of  6 variables:
##  $ Vandal         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Minor          : int  1 1 0 1 1 0 0 0 1 0 ...
##  $ LoggedIn       : int  1 1 1 0 1 1 1 1 1 0 ...
##  $ HTTP           : int  1 0 0 0 1 0 0 0 1 0 ...
##  $ NumWordsAdded  : int  96 3 0 10 94 4 3 2 5 0 ...
##  $ NumWordsRemoved: int  0 1 4 92 10 4 3 0 2 0 ...
```

```
View(Wiki)

#number of cases detected = 1815
Number_of_cases_detected <- sum(Wiki$Vandal)
```

Second, the average number of words added is about 4.05 words, while the average number of words removed is about 3.5 words.

```
#finding the mean for column NumWordsAdded and NumWordsRemoved
Mean.word.added <- mean(Wiki$NumWordsAdded)
Mean.word.rmv <- mean(Wiki$NumWordsRemoved)
```

Let’s explore the correlation between variables for a bit. From the correlation outputs below, we can see that variable “LoggedIn” is most correlated with variable Vandal as their correlation coefficient is about -0.43.

```
cor(Wiki)
```

```
##                        Vandal        Minor    LoggedIn        HTTP
## Vandal           1.0000000000 -0.213995217 -0.42925457  0.15155368
## Minor           -0.2139952169  1.000000000  0.44516561 -0.08429685
## LoggedIn        -0.4292545749  0.445165610  1.00000000 -0.11063301
## HTTP             0.1515536849 -0.084296852 -0.11063301  1.00000000
## NumWordsAdded   -0.0007289019 -0.007726385  0.02622296  0.11442149
## NumWordsRemoved  0.0363597359 -0.037629294 -0.03642207 -0.03986582
##                 NumWordsAdded NumWordsRemoved
## Vandal          -0.0007289019      0.03635974
## Minor           -0.0077263847     -0.03762929
## LoggedIn         0.0262229639     -0.03642207
## HTTP             0.1144214902     -0.03986582
## NumWordsAdded    1.0000000000      0.02523534
## NumWordsRemoved  0.0252353411      1.00000000
```

Step 2: Now let's randomly split the data into a training set and a testing set. In this case, I'm leaving 70% of the data in the training set. Let's see would would be the accuracy on the testing set of a simple baseline method that always predicts "not vandalism" for every edit.

```
#First, let's install all the useful packages for partioning data and creating classification mo
dels.
#install.packages("caTools")
library(caTools)
#install.packages("rpart")
library(rpart)
#install.packages("rpart.plot")
library(rpart.plot)

#Before splitting the data, set seed so we can make sure that all our random partitions of our s
ingle dataset Wiki into training and testing data will be the same. (to replicate random result
s)

library(dplyr)
```

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

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

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

```
#Replace all N/A with 0
Wiki %>% mutate_all(~replace(., is.na(.), 0))
```

Vandal <dbl>	Minor <dbl>	LoggedIn <dbl>	HTTP <dbl>	NumWordsAdded <dbl>	NumWordsRemoved <dbl>
0	1	1	1	96	0
0	1	1	0	3	1
0	0	1	0	0	4
0	1	0	0	10	92
0	1	1	1	94	10
0	0	1	0	4	4
0	0	1	0	3	3
0	0	1	0	2	0
0	1	1	1	5	2
0	0	0	0	0	0

1-10 of 3,876 rows Previous **1** 2 3 4 5 6 … 388 Next

```
#Convert data types to factor for validations later on.

Wiki$Vandal <- as.factor(Wiki$Vandal)
Wiki$Minor <- as.factor(Wiki$Minor)
Wiki$LoggedIn <- as.factor(Wiki$LoggedIn)
Wiki$HTTP <- as.factor(Wiki$HTTP)
Wiki$NumWordsAdded <- as.factor(Wiki$NumWordsAdded)
Wiki$NumWordsRemoved <- as.factor(Wiki$NumWordsRemoved)

#set seed for reproducible splitting outcomes
set.seed(1234)

#Now split the data, randomly assigning 70% to training and 30% to testing.
Split = sample.split(Wiki$Vandal, SplitRatio = 0.70)

Wiki.Train <- subset(Wiki, Split == TRUE)
Wiki.Test <- subset(Wiki, Split == FALSE)
```

Here, I Built a CART model as a baseline to predict Vandal, using all of the other variables as independent variables.

```
#Now we need to tune the CART model using caret package. Here we used a 10 fold valiadation to f
ind the best parameter for our data.

library(caret)
```

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

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

```
#Here's how I decided on the number of folds for the validation: K = N/N*(%size of testing test)
where N = Size of data set and K = Fold. In our case, we have 3876 rows of data and the testing
 set is 30%, so we can use about 3 folds forour CART1 cross validation. (3876/(0.3*3876))
#install.packages("MLmetrics")
library("MLmetrics")
```

```
## Warning: package 'MLmetrics' was built under R version 4.0.4
```

```
##
## Attaching package: 'MLmetrics'
```

```
## The following objects are masked from 'package:caret':
##
##     MAE, RMSE
```

```
## The following object is masked from 'package:base':
##
##     Recall
```

```
Control=trainControl(method= "repeatedcv",number=3,repeats=3,classProbs=TRUE,summaryFunction=mul
tiClassSummary)
cart_baseline=caret::train(make.names(Vandal)~.,data=Wiki.Train,method="rpart",trControl=Contro
l,tuneLength=3)
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

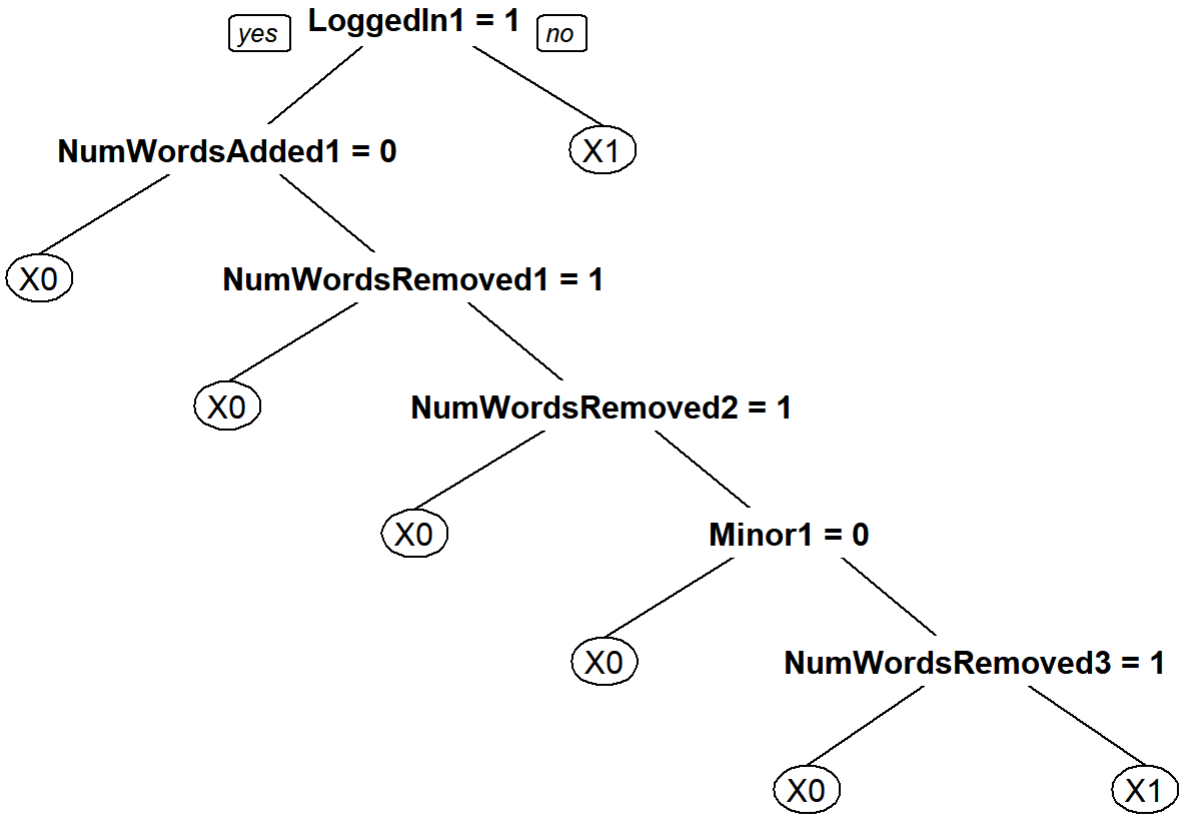
```
cart_baseline
```

```
## CART
##
## 2713 samples
##    5 predictor
##    2 classes: 'X0', 'X1'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 3 times)
## Summary of sample sizes: 1809, 1808, 1809, 1808, 1809, 1809, ...
## Resampling results across tuning parameters:
##
##   cp           logLoss    AUC        prAUC      Accuracy   Kappa      F1
##   0.002755906  0.5968336  0.7141309  0.3789893  0.7101600  0.4102249  0.7513570
##   0.003346457  0.5976594  0.7103390  0.3809605  0.7092996  0.4082280  0.7512980
##   0.380314961  0.6652720  0.5618880  0.0634703  0.5864350  0.1259541  0.7115905
##   Sensitivity  Specificity  Pos_Pred_Value  Neg_Pred_Value  Precision
##   0.8237468    0.5811161    0.6910864       0.7447224       0.6910864
##   0.8260568    0.5766506    0.6893681       0.7459203       0.6893681
##   0.9468699    0.1769060    0.5781008       0.7461827       0.5781008
##   Recall     Detection_Rate  Balanced_Accuracy
##   0.8237468  0.4381335       0.7024315
##   0.8260568  0.4393626       0.7013537
##   0.9468699  0.5036232       0.5618880
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.002755906.
```

Accuracy was used to select the optimal model using the largest value. As a result, 0.002755906 was selected as the optimal cp value to test the model.

Let’s plot the CART tree. Seems like the ‘LoggedIn’, ‘NumWordsA’, and ‘NumWordsR’ are selected by the model to be the top predictors.

```
par(mar=c(1,1,1,1))
prp(cart_baseline$finalModel)
```



Let's check the accuracy of the model on the test set.

```
#Predict the values of Vandal using the testing data set as input of our trained model WikiTree.
Note that the default setting of the predict function below assumes a 0.5 threshold and use 0.5
 as a probability benchmark to predict 0 or 1 scores.

cart_baseline.Predicts <- predict(cart_baseline, newdata = Wiki.Test, type = "raw")


#After we got the predictions, let's compare the predicted values to the actual values.
Comparison.Table <- table(Wiki.Test$Vandal, cart_baseline.Predicts)
Comparison.Table
```

```
##    cart_baseline.Predicts
##      X0  X1
##   0 505 113
##   1 221 324
```

```
WikiTree.Accuracy <- sum(diag(Comparison.Table))/sum(Comparison.Table)
WikiTree.Accuracy
```

```
## [1] 0.7128117
```

In the confusion matrix above, we can see that the model accuracy is 0.7128117, meaning our model accurately predicted about 71.28% of all vandalism on the page.

Now we have a basline CART model, let's build a more sophisticated random forest model to predict "Vandal", using all the other variables as independent variables.

```
#install.packages("randomForest")
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```
##
## Attaching package: 'randomForest'
```

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

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

```
#install.packages("caret")
library(caret)

#Convert the independent variables(Training & Testing) from factors back to integers
Wiki.Train$Minor <- as.integer(Wiki.Train$Minor)
Wiki.Train$LoggedIn <- as.integer(Wiki.Train$LoggedIn)
Wiki.Train$HTTP <- as.integer(Wiki.Train$HTTP)
Wiki.Train$NumWordsAdded <- as.integer(Wiki.Train$NumWordsAdded)
Wiki.Train$NumWordsRemoved <- as.integer(Wiki.Train$NumWordsRemoved)

Wiki.Test$Minor <- as.integer(Wiki.Test$Minor)
Wiki.Test$LoggedIn <- as.integer(Wiki.Test$LoggedIn)
Wiki.Test$HTTP <- as.integer(Wiki.Test$HTTP)
Wiki.Test$NumWordsAdded <- as.integer(Wiki.Test$NumWordsAdded)
Wiki.Test$NumWordsRemoved <- as.integer(Wiki.Test$NumWordsRemoved)

#Find the right number of mtry
Wiki.RF.values <- vector(length=5)
for(i in 1:5) {
  Wiki.RF <- randomForest(Vandal ~ ., data = Wiki.Train, mtry = i, ntree = 500)
  Wiki.RF.values[i] <- Wiki.RF$err.rate[nrow(Wiki.RF$err.rate),1]
}

Wiki.RF.values
```

```
## [1] 0.2782897 0.2635459 0.2642831 0.2720236 0.2727608
```

```
# As we can see from above, the 3rd value correspond to metry = 3has the lowest out of bag error rate. So we can just got with mtry = 3 in this model.

Wiki.RF <- randomForest(Vandal ~., data = Wiki.Train, ntree=500, mtry = 3, proximity = TRUE)
Wiki.RF
```

```
##
## Call:
##  randomForest(formula = Vandal ~ ., data = Wiki.Train, ntree = 500,      mtry = 3, proximity
= TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 26.5%
## Confusion matrix:
##      0   1 class.error
## 0 1281 162   0.1122661
## 1  557 713   0.4385827
```

```
Wiki.RF.Predict <- predict(Wiki.RF, newdata = Wiki.Test, type = "class")

#After we got the predictions, let's compare the predicted values to the actual values.
RF.Comparison.Table <- table(Wiki.Test$Vandal, Wiki.RF.Predict)
Comparison.Table
```

```
##    cart_baseline.Predicts
##      X0  X1
##   0 505 113
##   1 221 324
```

```
Wiki.RF.Accuracy <- sum(diag(RF.Comparison.Table))/sum(RF.Comparison.Table)
Wiki.RF.Accuracy
```

```
## [1] 0.7394669
```

Our random forest model accuracy turns out to be 73.68%, which is slightly better than our CART model(71.28% accuracy) by 2.4%.

Business Implications:

Both of the models I built would be useful as baseline models for Wikipedia to detect vandalism. The model accuracy can be improved with the addition of more sophisticated techniques to optimize the model robustness. However, as the baseline models, the two models I built above should suffice as bench marks for Wikipedia.

If I could collect more data about the edits, I would want to use variables such as 1)the associated editor's edit reversal records, and 2)the number of historical edits happened to the page.

I wanted the variables mentioned above because 1) if the associated editor's edits to the page has been reversed multiple times, that associated editors is likely to be a habitual vandals. This can be a strong indicator of whether the edits would likely to be vandalism or not. 2) The number of historical edits for the page could be an indicator as well because if there's a high number of historical edits, there might be higher number of vandalism happen to the page because it's well known.