Ensemble Learning and Text Mining

Blake Pappas

2023-12-17

```r
library(dplyr)
library(caret)
library(rpart)
library(e1071)
library(fastAdaboost)
library(ipred)
library(randomForest)
library(tm)
library(wordcloud)
```

Part I. Ensemble Learning in R

Load the "bank_small.csv" data file:

```r
bank = read.csv("bank_small.csv", stringsAsFactors = TRUE)

train_rows = createDataPartition(y = bank$y,
                                 p = 0.70, list = FALSE)

bank_train = bank[train_rows, ]
bank_test = bank[-train_rows, ]
```

Bagging Using Training/Testing Split

```r
bag_model = bagging(y ~ ., data = bank_train, nbagg = 50)

# Make Predictions and Evaluate Performance
pred = predict(bag_model, bank_test)
confusionMatrix(pred, bank_test$y, mode = "prec_recall", positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1278  108
```

```
##         yes   46   67
##
##                Accuracy : 0.8973
##                  95% CI : (0.8808, 0.9122)
##     No Information Rate : 0.8833
##     P-Value [Acc > NIR] : 0.04768
##
##                   Kappa : 0.4114
##
##  Mcnemar's Test P-Value : 8.855e-07
##
##               Precision : 0.59292
##                  Recall : 0.38286
##                      F1 : 0.46528
##              Prevalence : 0.11674
##          Detection Rate : 0.04470
##    Detection Prevalence : 0.07538
##       Balanced Accuracy : 0.67406
##
##        'Positive' Class : yes
##
```

Bagging Using Cross Validation and For Loop

```r
# Create the Folds
cv = createFolds(bank$y, k = 5)

# Make a Vector to Store Model Type, Number of Bagging Models, and F1 from Each Fold
bagging_models = vector()
F_Measure = vector()

for(i in seq(50, 250, 50)) {
  F1 = vector()

  # Loop Through Each Fold
  for(test_rows in cv) {
    bank_train = bank[-test_rows, ]
    bank_test = bank[test_rows, ]

    # Train the Model and Evaluate Performance
    bag_model = bagging(y ~ ., data = bank_train, nbagg = i)
    pred = predict(bag_model, bank_test)
    cm = confusionMatrix(pred, bank_test$y, mode = "prec_recall", positive = "yes")

    # Add the F1 of the Current Fold
    F1 = append(F1, cm$byClass[7])
  }
  bagging_models = append(bagging_models, i)
  F_Measure = append(F_Measure, mean(F1))
}

data.frame(Bagging_Models = bagging_models, F_Measure = F_Measure)
```

```
##   Bagging_Models F_Measure
## 1             50 0.4402639
## 2            100 0.4540431
## 3            150 0.4654844
## 4            200 0.4640037
## 5            250 0.4521020
```

Boosting Using Training/Testing Split

```r
adaboost_model = adaboost(y ~ ., data = bank_train, nIter = 100)

# Make Predictions and Evaluate Performance
pred = predict(adaboost_model, bank_test)
confusionMatrix(pred$class, bank_test$y, mode = "prec_recall", positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  854  78
##        yes  29  39
##
##                Accuracy : 0.893
##                  95% CI : (0.8722, 0.9115)
##     No Information Rate : 0.883
##     P-Value [Acc > NIR] : 0.1754
##
##                   Kappa : 0.3672
##
##  Mcnemar's Test P-Value : 3.478e-06
##
##               Precision : 0.5735
##                  Recall : 0.3333
##                      F1 : 0.4216
##              Prevalence : 0.1170
##          Detection Rate : 0.0390
##    Detection Prevalence : 0.0680
##       Balanced Accuracy : 0.6502
##
##        'Positive' Class : yes
##
```

Boosting Using Cross Validation and For Loop

```r
# Create the Folds
cv = createFolds(bank$y, k = 5)

# Make a Vector to Store Model Type, Number of Iterations, and F1 from Each Fold
iterations = vector()
F_Measure = vector()
```

```r
for(i in seq(50, 250, 50)) {
  F1 = vector()

  # Loop Through Each Fold
  for(test_rows in cv) {
    bank_train = bank[-test_rows, ]
    bank_test = bank[test_rows, ]

    # Train the Model and Evaluate Performance
    adaboost_model = adaboost(y ~ ., data = bank_train, nIter = i)
    pred = predict(adaboost_model, bank_test)
    cm = confusionMatrix(pred$class, bank_test$y, mode = "prec_recall", positive = "yes")

    # Add the F1 of the Current Fold
    F1 = append(F1, cm$byClass[7])
  }
  iterations = append(iterations, i)
  F_Measure = append(F_Measure, mean(F1))
}

data.frame(Iterations = iterations, F_Measure = F_Measure)
```

```
##   Iterations F_Measure
## 1         50 0.4468141
## 2        100 0.4227642
## 3        150 0.4223755
## 4        200 0.4179687
## 5        250 0.4125495
```

Random Forest Using Training/Testing Split

```r
rf_model = randomForest(y ~ ., data = bank_train, ntree = 100)

# Make Predictions and Evaluate Performance
pred = predict(rf_model, bank_test)
confusionMatrix(pred, bank_test$y, mode = "prec_recall", positive = "yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  849  74
##        yes  34  43
##
##                Accuracy : 0.892
##                  95% CI : (0.8711, 0.9106)
##     No Information Rate : 0.883
##     P-Value [Acc > NIR] : 0.2025466
##
##                   Kappa : 0.3863
```

```
##
##  Mcnemar's Test P-Value : 0.0001749
##
##               Precision : 0.5584
##                  Recall : 0.3675
##                      F1 : 0.4433
##              Prevalence : 0.1170
##          Detection Rate : 0.0430
##    Detection Prevalence : 0.0770
##       Balanced Accuracy : 0.6645
##
##        'Positive' Class : yes
##
```

Random Forest Using Cross Validation and For Loop

```r
# Create the Folds
cv = createFolds(bank$y, k = 5)

# Make a Vector to Store Model Type, Number of Iterations, and F1 from Each Fold
trees = vector()
F_Measure = vector()

for(i in seq(50, 250, 50)) {
  F1 = vector()

  # Loop Through Each Fold
  for(test_rows in cv) {
    bank_train = bank[-test_rows, ]
    bank_test = bank[test_rows, ]

    # Train the Model and Evaluate Performance
    rf_model = randomForest(y ~ ., data = bank_train, ntree = i)
    pred = predict(rf_model, bank_test)
    cm = confusionMatrix(pred, bank_test$y, mode = "prec_recall", positive = "yes")

    # Add the F1 of the Current Fold
    F1 = append(F1, cm$byClass[7])
  }
  trees = append(trees, i)
  F_Measure = append(F_Measure, mean(F1))
}

data.frame(Trees = trees, F_Measure = F_Measure)
```

```
##   Trees F_Measure
## 1    50 0.4709258
## 2   100 0.4624959
## 3   150 0.4563831
## 4   200 0.4758236
## 5   250 0.4675108
```

Stacking Using Training/Testing Split

```r
# Build a Stacking Model

train_rows = createDataPartition(y = bank$y,
                                  p = 0.50, list = FALSE)

bank_train = bank[train_rows, ]
bank_test = bank[-train_rows, ]


# Train Three Base Learners
tree_model = rpart(y ~ ., data = bank_train, method = "class",
                   parms = list(split = "information"))

logit_model = glm(y ~ ., data = bank_train,
                  family = binomial(link = "logit"))

svm_model = svm(y ~ ., data = bank_train,
                kernel = "polynomial", degree = 2)

# Make Predictions Using Base Learners
pred_tree = predict(tree_model, bank_test, type = "class")

pred_logit_prob = predict(logit_model, bank_test, type = "response")

pred_logit = ifelse(pred_logit_prob > 0.5, "yes", "no")

pred_svm = predict(svm_model, bank_test)

# Add Base Learners' Predictions to the bank_test Data
bank_test = bank_test %>%
  mutate(pred_tree = pred_tree,
         pred_logit = pred_logit,
         pred_svm = pred_svm)

# Do the Second Split
train2_rows = createDataPartition(y = bank_test$y,
                                  p = 0.50, list = FALSE)

bank_train2 = bank_test[train2_rows, ]
bank_test2 = bank_test[-train2_rows, ]

# Build the Naive Bayes Combiner
nb_model = naiveBayes(y ~ ., data = bank_train2)

# Evaluate the Naive Bayes Model
pred_nb = predict(nb_model, bank_test2)

confusionMatrix(factor(pred_nb), bank_test2$y,
                mode = "prec_recall", positive = "yes")
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction   no  yes
##        no  1003   64
##        yes  100   82
##
##                Accuracy : 0.8687
##                  95% CI : (0.8487, 0.8869)
##     No Information Rate : 0.8831
##     P-Value [Acc > NIR] : 0.946509
##
##                   Kappa : 0.4255
##
##  Mcnemar's Test P-Value : 0.006275
##
##               Precision : 0.45055
##                  Recall : 0.56164
##                      F1 : 0.50000
##              Prevalence : 0.11689
##          Detection Rate : 0.06565
##    Detection Prevalence : 0.14572
##       Balanced Accuracy : 0.73549
##
##        'Positive' Class : yes
##
```

Part II. Text Mining in R

1. Import the "FB Posts.csv" data file into R. Then, convert the text into a corpus.

```r
fb_posts_text = read.csv("FB Posts.csv")

fb_posts_corpus = Corpus(VectorSource(fb_posts_text$Text))
```

2. Print out the contents of the 100th post

```r
fb_posts_corpus[[100]]$content
```

```
## [1] "luv southwest"
```

3. Remove all punctuations from the posts

```
fb_posts_corpus = tm_map(fb_posts_corpus, removePunctuation)
```

4. Convert all text to lowercase

```
fb_posts_corpus = tm_map(fb_posts_corpus, tolower)
```

5. Remove English stopwords

```
fb_posts_corpus = tm_map(fb_posts_corpus, removeWords,
                          stopwords('english'))
```

6. Perform Stemming

```
fb_posts_corpus = tm_map(fb_posts_corpus, stemDocument)
```

7. Obtain the TF-IDF matrix of the corpus

```
dtm = DocumentTermMatrix(fb_posts_corpus)

dtm_matrix = as.matrix(dtm)
```

8. Report the top 10 most common words in the corpus

```
word_freq = colSums(as.matrix(dtm))

word_freq_sorted = sort(word_freq, decreasing = TRUE)

word_freq_sorted[1:10]
```

```
##   love    get custom servic   just   call   like   time    now flight
##    205    187    168    161    161    158    153    146    140    133
```

9. Make a WordCloud. Set the minimum word frequency to be 5.

```r
wordcloud(words = names(word_freq_sorted),
          freq = word_freq_sorted,
          min.freq = 5)
```