REPORT

수강과목	:	빅데이터통계분석
담당교수	:	선호근
학 과	:	통계학과
학 번	:	201611531
이 름	:	정호재
제출일자	:	2020.11.19.

Homework Assignment 04

The Due Date : By Thursday, November, 19th in class
Your solution should include R codes and the answer of each question.
You need to upload your R codes on http://plato.pusan.ac.kr for full credits.
You may collaborate on this problem but you must write up your own solution.

Open the data set `College` in the `R` package 'ISLR'. The data information is available with `?College`. All of 17 variables except the response variable `Private` are used for predictors. We randomly generate 600 training samples and 177 test samples. This random generation repeats 100 times such that

```
> RNGkind(sample.kind = "Rounding")
> set.seed(1111)
> id <- rep(c("tr", "te"), c(600, 177))
> index <- matrix(id, length(id), 100)
> index <- apply(index, 2, sample)
```

In the `i`-th replicate, `"tr"` and `"te"` of `index[,i]` stand for 600 training samples and 177 test samples, respectively.

1. Develop an `R` function to compute accuracy (ACC), Matthews correlation coefficient (MCC), and F1 score (F_1), where two input variables are predicted class labels of 177 test samples, and real class labels of the test set. e.g., `Private=="Yes"` or `Private=="No"`. Note that ACC is equivalent to "1−misclassification rate" and the formula of MCC and F_1 are available in your Homework Assignment 3. To answer the following questions (Q2 $\sim$ Q6), ACC, MCC, and F_1 should be first computed each replicate and then they should be averaged over 100 different replicates.

```
fun.acc <- function(pred, test){
  acc <- NA
  tp <- sum(pred[test=="Yes"] == "Yes")
  tn <- sum(pred[test=="No"] == "No")
  fp <- sum(pred=="Yes") - tp
  fn <- sum(test=="Yes") - tp
  acc <- (tp+tn)/(tp+tn+fp+fn)

  return(acc)
}

fun.mcc <- function(pred, test){
  mcc <- NA
  tp <- sum(pred[test=="Yes"] == "Yes")
  tn <- sum(pred[test=="No"] == "No")
  fp <- sum(pred=="Yes") - tp
  fn <- sum(test=="Yes") - tp
  mcc <- ((tp*tn)-(fp*fn))/(sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn)))

  return(mcc)
}
```

```
fun.f1 <- function(pred, test){
  f1 <- NA
  tp <- sum(pred[test=="Yes"] == "Yes")
  tn <- sum(pred[test=="No"] == "No")
  fp <- sum(pred=="Yes") - tp
  fn <- sum(test=="Yes") - tp
  f1 <- 2*tp/(2*tp+fp+fn)

  return(f1)
}
```

2. For each replicate, build 3 classification models such as LDA(Linear Discriminant Analysis), QDA(Quadratic Discriminant Analysis) and NB(Naive Bayes) for the training set, and then predict the class labels of the test set. Compute averaged ACC, MCC, and F_1 of the test set, using three models. (Just use a threshold of 0.5 for classification).

	ACC	MCC	F1
LDA	0.9368927	0.8374779	0.9572309
QDA	0.9025989	0.7444609	0.9352927
NB	0.9056497	0.7531308	0.9370299

3. For each replicate, build a classification tree for the training set, and then predict the class labels of the test set. Compute averaged ACC, MCC, and F_1 of the test set, using the estimated tree. In this question, do not prune the tree.

	ACC	MCC	F1
Classification Tree	0.9198305	0.7974461	0.9447906

4. For each replicate, perform 10-fold cross validation (CV) for the training set to find the optimal tree size. You must use the following group id matrix `cv.index` to separate 10 groups for each replicate.

```
> RNGkind(sample.kind = "Rounding")
> set.seed(4444)
> cv.id <- rep(seq(10), 60)
> cv.index <- matrix(cv.id, length(cv.id), 100)
> cv.index <- apply(cv.index, 2, sample)
```

The function `cv.tree(..., rand=cv.index[,i], FUN=prune.misclass)` should be used for 10-fold CV for the `i`-th replicate. If you have the exactly same deviance values for different sizes of trees, you should pick up the largest tree for your optimal tree. Compute averaged ACC, MCC, and F_1 of the test set, using the optimal tree.

	ACC	MCC	F1
Cross Validated Tree	0.9190395	0.7953039	0.9443112

5. For each replicate, build a classification tree for the training set using the bagging method. For each replicate, 300 bootstrap samples for the training set can be generated by

```
> RNGkind(sample.kind = "Rounding")
> set.seed(5555)
> bt.index <- array(0, c(600, 300, 100))
> for (t in 1:100) {
+   for (b in 1:300) {
+       u <- unique(sample(1:600, replace=TRUE))
+       bt.index[u, b, t] <- 1
+   }
+ }
```

For the `i`-th replicate, the matrix `bt.index[,,i]` consists of either `1` or `0` for 600 training samples and 300 bootstrap replications, where `1` stands for '**selected**' and `0` stands for '**not selected**' in the bootstrap replicate. The prediction of the bagging tree should follow the majority vote rule. For example, if the jth test observation obtains at least 150 votes for `Private=="Yes"` among 300 classification trees (generated by the bootstrap replications), the predicted class label of the jth test observation is `"Yes"`; otherwise `"No"`. Compute averaged ACC, MCC, and F_1 of the test set, using the bagging tree.

	ACC	MCC	F1
Bagging Tree	0.9355932	0.835584	0.9559561

6. For each replicate, build random forest for the training set where the number of predictors (m) is considered as 1, 2, 3, 4 ,5 or 6. To find the optimal value of m, perform 10-fold cross validation (CV) using `cv.index` in Q4. The optimal value of m should minimize an averaged misclassification rate of 10 folds. If multiple m values have the exactly same misclassification rate, you must select a smaller value of m. In order to fit random forest, use the `randomForest` function with `ntree=1000`. If you find the optimal value of m each replicate, compute ACC, MCC, and F_1 of the test set using random forest with the optimal m. Finally, provide the averaged ACC, MCC, and F_1 over 100 replicates.

	ACC	MCC	F1
Random Forest	0.940565	0.8471585	0.9595786

7. Summarize your result using the following table

	ACC	MCC	F_1
Q2 (LDA)			
Q2 (QDA)			
Q2 (NB)			
Q3 (Tree)			
Q4 (Cross-validated tree)			
Q5 (Bagging)			
Q6 (Random Forest)			

Who is winner?

	ACC	MCC	F1
LDA	0.9368927	0.8374779	0.9572309
QDA	0.9025989	0.7444609	0.9352927
NB	0.9056497	0.7531308	0.9370299
Classification Tree	0.9198305	0.7974461	0.9447906
Cross Validated Tree	0.9190395	0.7953039	0.9443112
Bagging Tree	0.9355932	0.8355840	0.9559561
Random Forest	**0.9405650**	**0.8471585**	**0.9595786**

Winner is Random Forest