Data 622 :: HW#2

Vishal Arora

12/6/2020

Contents

PART-A

STEP#0 : Pick any two classifiers of (SVM,Logistic,DecisionTree,NaiveBayes). Pick heart or ecoli dataset. Heart is simpler and ecoli compounds the problem as it is NOT a balanced dataset. From a grading perspective both carry the same weight.
STEP#1 : For each classifier, Set a seed (43)
STEP#2 : Do a 80/20 split and determine the Accuracy, AUC and as many metrics as returned by the Caret package (confusionMatrix) Call this the base_metric. Note down as best as you can development (engineering) cost as well as computing cost(elapsed time).
Start with the original dataset and set a seed (43). Then run a cross validation of 5 and 10 of the model on the training set. Determine the same set of metrics and compare the cv_metrics with the base_metric. Note down as best as you can development (engineering) cost as well as computing cost(elapsed time).
Start with the original dataset and set a seed (43) Then run a bootstrap of 200 resamples and compute the same set of metrics and for each of the two classifiers build a three column table for each experiment (base, bootstrap, cross-validated). Note down as best as you can development (engineering) cost as well as computing cost(elapsed time).

Load Libraries

We will be using Logistic regression and Naive baiyes for Part 1.

```
path<-"C:\\CUNY_AUG27\\DATA622\\heart.csv"
heartDT<-read.csv(path,head=T,sep=',',stringsAsFactors=F)

#Overview of the data
head(heartDT)
```

```
##   ï..age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal
## 1     63   1  3      145  233   1       0     150     0     2.3     0  0    1
## 2     37   1  2      130  250   0       1     187     0     3.5     0  0    2
## 3     41   0  1      130  204   0       0     172     0     1.4     2  0    2
## 4     56   1  1      120  236   0       1     178     0     0.8     2  0    2
## 5     57   0  0      120  354   0       1     163     1     0.6     2  0    2
## 6     57   1  0      140  192   0       1     148     0     0.4     1  0    1
##   target
## 1      1
## 2      1
## 3      1
## 4      1
## 5      1
## 6      1
```

```r
dim(heartDT)
```

```
## [1] 303  14
```

```r
#changing the name of first column to age
names(heartDT)[[1]] <- "age"
names(heartDT)
```

```
##  [1] "age"      "sex"      "cp"       "trestbps" "chol"     "fbs"
##  [7] "restecg"  "thalach"  "exang"    "oldpeak"  "slope"    "ca"
## [13] "thal"     "target"
```

```r
#To check in NA/NaN values are there
sum(is.na(heartDT))
```

```
## [1] 0
```

```r
#skimr package is another good way to check descriptive statistics of data.

skimmed_data <- skim(heartDT)
View(skimmed_data)
heartDT$target <- as.factor(heartDT$target)
```

As clearly visible that the age variable is normally distributed. Hence, there is no bias in the data set used.

Splitting data & applying models

```r
set.seed(43)
trainidx<-sample(1:nrow(heartDT) , size=round(0.80*nrow(heartDT)),replace=F)

train_data <- heartDT[trainidx,]

test_data <- heartDT[-trainidx,]
```

Models

Logistic Regression

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 18  2
```

```
##          1  9 32
##
##                Accuracy : 0.8197
##                  95% CI : (0.7002, 0.9064)
##     No Information Rate : 0.5574
##     P-Value [Acc > NIR] : 1.469e-05
##
##                   Kappa : 0.6245
##
##  Mcnemar's Test P-Value : 0.07044
##
##             Sensitivity : 0.6667
##             Specificity : 0.9412
##          Pos Pred Value : 0.9000
##          Neg Pred Value : 0.7805
##              Prevalence : 0.4426
##          Detection Rate : 0.2951
##    Detection Prevalence : 0.3279
##       Balanced Accuracy : 0.8039
##
##        'Positive' Class : 0
##
##           Reference
## Prediction  0  1
##          0 18  2
##          1  9 32
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 18  2
##          1  9 32
##
##                Accuracy : 0.8197
##                  95% CI : (0.7002, 0.9064)
##     No Information Rate : 0.5574
##     P-Value [Acc > NIR] : 1.469e-05
##
##                   Kappa : 0.6245
##
##  Mcnemar's Test P-Value : 0.07044
##
##             Sensitivity : 0.6667
##             Specificity : 0.9412
##          Pos Pred Value : 0.9000
##          Neg Pred Value : 0.7805
##              Prevalence : 0.4426
##          Detection Rate : 0.2951
##    Detection Prevalence : 0.3279
##       Balanced Accuracy : 0.8039
##
##        'Positive' Class : 0
##
```

```
##           Reference
## Prediction  0  1
##          0 18  2
##          1  9 32
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 18  2
##          1  9 32
##
##                Accuracy : 0.8197
##                  95% CI : (0.7002, 0.9064)
##     No Information Rate : 0.5574
##     P-Value [Acc > NIR] : 1.469e-05
##
##                   Kappa : 0.6245
##
##  Mcnemar's Test P-Value : 0.07044
##
##             Sensitivity : 0.6667
##             Specificity : 0.9412
##          Pos Pred Value : 0.9000
##          Neg Pred Value : 0.7805
##              Prevalence : 0.4426
##          Detection Rate : 0.2951
##    Detection Prevalence : 0.3279
##       Balanced Accuracy : 0.8039
##
##        'Positive' Class : 0
##
##           Reference
## Prediction  0  1
##          0 18  2
##          1  9 32
```

SVM

```
## Support Vector Machines with Linear Kernel
##
## 242 samples
##  13 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 242, 242, 242, 242, 242, 242, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8181897  0.6298784
##
## Tuning parameter 'C' was held constant at a value of 1
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction  0  1
##          0 18  2
##          1  9 32
##
##                Accuracy : 0.8197
##                  95% CI : (0.7002, 0.9064)
##     No Information Rate : 0.5574
##     P-Value [Acc > NIR] : 1.469e-05
##
##                   Kappa : 0.6245
##
##  Mcnemar's Test P-Value : 0.07044
##
##             Sensitivity : 0.6667
##             Specificity : 0.9412
##          Pos Pred Value : 0.9000
##          Neg Pred Value : 0.7805
##              Prevalence : 0.4426
##          Detection Rate : 0.2951
##    Detection Prevalence : 0.3279
##       Balanced Accuracy : 0.8039
##
##        'Positive' Class : 0
##
```

```
## Support Vector Machines with Linear Kernel
##
## 242 samples
##  13 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 193, 194, 194, 194, 193
## Resampling results:
##
##   Accuracy   Kappa
##   0.8346088  0.6648322
##
## Tuning parameter 'C' was held constant at a value of 1
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 18  2
##          1  9 32
##
##                Accuracy : 0.8197
##                  95% CI : (0.7002, 0.9064)
##     No Information Rate : 0.5574
##     P-Value [Acc > NIR] : 1.469e-05
##
```

```
##                  Kappa : 0.6245
##
##  Mcnemar's Test P-Value : 0.07044
##
##             Sensitivity : 0.6667
##             Specificity : 0.9412
##          Pos Pred Value : 0.9000
##          Neg Pred Value : 0.7805
##              Prevalence : 0.4426
##          Detection Rate : 0.2951
##    Detection Prevalence : 0.3279
##       Balanced Accuracy : 0.8039
##
##        'Positive' Class : 0
##
```

```
## Support Vector Machines with Linear Kernel
##
## 242 samples
##  13 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 218, 218, 218, 218, 218, 218, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8431667  0.6806803
##
## Tuning parameter 'C' was held constant at a value of 1
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 18  2
##          1  9 32
##
##                Accuracy : 0.8197
##                  95% CI : (0.7002, 0.9064)
##     No Information Rate : 0.5574
##     P-Value [Acc > NIR] : 1.469e-05
##
##                   Kappa : 0.6245
##
##  Mcnemar's Test P-Value : 0.07044
##
##             Sensitivity : 0.6667
##             Specificity : 0.9412
##          Pos Pred Value : 0.9000
##          Neg Pred Value : 0.7805
##              Prevalence : 0.4426
##          Detection Rate : 0.2951
##    Detection Prevalence : 0.3279
```

```
##       Balanced Accuracy : 0.8039
##
##        'Positive' Class : 0
##
```

Bootstaping with 200 resamples

```
## Generalized Linear Model
##
## 303 samples
##  13 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (200 reps)
## Summary of sample sizes: 303, 303, 303, 303, 303, 303, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8188716  0.6325618
```

```
## Support Vector Machines with Linear Kernel
##
## 303 samples
##  13 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (200 reps)
## Summary of sample sizes: 303, 303, 303, 303, 303, 303, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.8194862  0.633356
##
## Tuning parameter 'C' was held constant at a value of 1
```

Result Matrix

ALGO	AUC	ACC	TPR	FPR	TNR	FNR	Computatio
Base Logistic metrics	0.8197	0.8039	0.6667	0.0588	0.9412	0.3333	0
CV5 Logistic metrics	0.8197	0.8039	0.6667	0.0588	0.9412	0.3333	0
CV10 Logistic metrics	0.8197	0.8039	0.6667	0.0588	0.9412	0.3333	0
Base SVM	0.8197	0.8039	0.6667	0.0588	0.9412	0.3333	0
CV5 SVM	0.8197	0.8039	0.6667	0.0588	0.9412	0.3333	0.01
CV10 SVM	0.8197	0.8039	0.6667	0.0588	0.9412	0.3333	0
LR bootstraping	0.8189	NA	NA	NA	NA	NA	3.9
SVM boostraping	0.8195	NA	NA	NA	NA	NA	3.36

Part B

Random Forest

Creating a baseline for comparison by using the recommend defaults for each parameter and mtry=floor(sqrt(ncol(x)))

```r
#
# Create model with default paramters
timer <- proc.time()
control <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "Accuracy"
set.seed(43)
mtry <- sqrt(ncol(train_data))
tunegrid <- expand.grid(.mtry=mtry)
rf_default <- train(target~., data=heartDT, method="rf", metric=metric, tuneGrid=tunegrid, trControl=co
totalTimeDelay <- (proc.time()- timer)[[3]]

print(paste0("Total time delay for default random forest :",totalTimeDelay))
```

```
## [1] "Total time delay for default random forest :7.38"
```

```r
print(rf_default)
```

```
## Random Forest
##
## 303 samples
##  13 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 273, 272, 273, 273, 273, 273, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.822991  0.6406402
##
## Tuning parameter 'mtry' was held constant at a value of 3.741657
```

Random Search Below model will generate 15 random values of mtry at each time tunning. We have 15 values because of tunning length is 15.

```r
timer <- proc.time()
control <- trainControl(method="repeatedcv", number=10, repeats=3, search="random")
ntree <- 3
set.seed(43)
#Random generate 15 mtry values with tuneLength = 15
mtry <- sqrt(ncol(train_data))
rf_random <- train(target~., data=heartDT, method="rf", metric=metric, tuneLength=15, trControl=control
totalTimeDelay <- (proc.time()- timer)[[3]]
print(paste0("Total time delay for default random forest :",totalTimeDelay))
```

```
## [1] "Total time delay for default random forest :62.74"
```

```r
print(rf_default)
```

```
## Random Forest
##
## 303 samples
##  13 predictor
##   2 classes: '0', '1'
##
```

```
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 273, 272, 273, 273, 273, 273, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.822991  0.6406402
##
## Tuning parameter 'mtry' was held constant at a value of 3.741657
```

Grid Search Create control function for training with 10 folds and keep 3 folds for training. search method is grid.

```
set.seed(43)
timer <- proc.time()
control <- trainControl(method='repeatedcv',
                        number=10,
                        repeats=3,
                        search='grid')
#create tunegrid with 15 values from 1:15 for mtry to tunning model. Our train function will change num
tunegrid <- expand.grid(.mtry = (1:15))

rf_gridsearch <- train(target ~ .,
                       data = heartDT,
                       method = 'rf',
                       metric = 'Accuracy',
                       tuneGrid = tunegrid)
totalTimeDelay <- (proc.time()- timer)[[3]]

print(paste0("Total time delay for default random forest :",totalTimeDelay))
```

```
## [1] "Total time delay for default random forest :85.7"
```

```
print(rf_gridsearch)
```

```
## Random Forest
##
## 303 samples
##  13 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 303, 303, 303, 303, 303, 303, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    1    0.8314587  0.6554028
##    2    0.8280021  0.6486340
##    3    0.8251322  0.6429775
##    4    0.8194193  0.6310565
##    5    0.8192520  0.6307234
##    6    0.8194304  0.6313800
##    7    0.8146500  0.6217703
##    8    0.8113603  0.6154098
```

```
##    9    0.8061321  0.6045047
##   10    0.8075789  0.6079520
##   11    0.8046959  0.6021528
##   12    0.8009863  0.5950569
##   13    0.7994504  0.5920460
##   14    0.8009242  0.5949481
##   15    0.8006197  0.5939232
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 1.
```

Manual tunning approach create many model caret scenarios with different manual parameters and compare its accuracy. We do this to evaluate different ntree while hodling mtry constant.

```
set.seed(43)
timer <- proc.time()
control <- trainControl(method = 'repeatedcv',
                        number = 10,
                        repeats = 3,
                        search = 'grid')
#create tunegrid
tunegrid <- expand.grid(.mtry = c(sqrt(ncol(train_data))))
modellist <- list()

#train with different ntree parameters
for (ntree in c(1000,1500,2000,2500)){

  fit <- train(target~.,
               data = heartDT,
               method = 'rf',
               metric = 'Accuracy',
               tuneGrid = tunegrid,
               trControl = control,
               ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
totalTimeDelay <- (proc.time()- timer)[[3]]
#Compare results
results <- resamples(modellist)

print(paste0("Total time delay for default random forest :",totalTimeDelay))
```

```
## [1] "Total time delay for default random forest :86.73"
```

```
print(summary(results))
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: 1000, 1500, 2000, 2500
## Number of resamples: 30
##
## Accuracy
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
```

```
## 1000 0.6666667 0.7806452 0.8387097 0.8208763 0.8666667 0.9333333    0
## 1500 0.7096774 0.7666667 0.8170189 0.8193981 0.8596774 0.9677419    0
## 2000 0.7000000 0.7741935 0.8360215 0.8261562 0.8666667 0.9354839    0
## 2500 0.7096774 0.7789210 0.8304598 0.8219960 0.8596774 0.9666667    0
##
## Kappa
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 1000 0.3181818 0.5522330 0.6723044 0.6366303 0.7291155 0.8617512    0
## 1500 0.4101480 0.5205479 0.6312936 0.6329136 0.7182094 0.9344609    0
## 2000 0.3720930 0.5383903 0.6658857 0.6471586 0.7339207 0.8697479    0
## 2500 0.4247423 0.5478123 0.6549932 0.6390540 0.7078347 0.9327354    0
```

Part C

Conclusion :

As is clearly visible from the result matrix for Part A , the Accuracy and other performance indicators are nearly same for the Base Model and CV Models for Logistic Regression & SVM. But it is the total time delay which is different for all the models. Between CV and Bootstraping models interms of Accuracy nothing much to choose for but it is the Total time taken by Bootstraping models which makes them more expensive in terms of computing resources.

Pareto's rule was implemented by implementing 80/20 rule while splitting the data and also the 20 % of our data is more critical in testing that our models are not overfitting or underfitting.

Occam's razor is the principle that, of two explanations that account for all the facts, the simpler one is more likely to be correct. In out case as Accuracy is not significant in case of Part A models and in terms of Total time delay also the difference is in seconds/milliseconds . So as per Ocam's razor principle we should go with Base Models (LR or SVM).

However Random Forest model with grid search has the best Accuracy with 83% , but if we tune our models manually then the accuracy can vary between 71% to 97% .But Random Forest model are very time consuming and require lot of computational power.