Lab7_May7

Aicha

2023-05-07

First we will import the dataset. The dataset is about two brands of orange juice.

```
library(ggplot2)
library(caret)

## Warning: package 'caret' was built under R version 4.2.3

## Loading required package: lattice

orange <-
read.csv('https://raw.githubusercontent.com/selva86/datasets/master/orange_ju
ice_withmissing.csv')
str(orange)

## 'data.frame':    1070 obs. of  18 variables:
##  $ Purchase      : chr  "CH" "CH" "CH" "MM" ...
##  $ WeekofPurchase: int  237 239 245 227 228 230 232 234 235 238 ...
##  $ StoreID       : int  1 1 1 1 7 7 7 7 7 7 ...
##  $ PriceCH       : num  1.75 1.75 1.86 1.69 1.69 1.69 1.69 1.75 1.75 1.75
...
##  $ PriceMM       : num  1.99 1.99 2.09 1.69 1.69 1.99 1.99 1.99 1.99 1.99
...
##  $ DiscCH        : num  0 0 0.17 0 0 0 0 0 0 0 ...
##  $ DiscMM        : num  0 0.3 0 0 0 0 0.4 0.4 0.4 0.4 ...
##  $ SpecialCH     : int  0 0 0 0 0 0 1 1 0 0 ...
##  $ SpecialMM     : int  0 1 0 0 0 1 1 0 0 0 ...
##  $ LoyalCH       : num  0.5 0.6 0.68 0.4 0.957 ...
##  $ SalePriceMM   : num  1.99 1.69 2.09 1.69 1.69 1.99 1.59 1.59 1.59 1.59
...
##  $ SalePriceCH   : num  1.75 1.75 1.69 1.69 1.69 1.69 1.69 1.75 1.75 1.75
...
##  $ PriceDiff     : num  0.24 -0.06 0.4 0 0 0.3 -0.1 -0.16 -0.16 -0.16 ...
##  $ Store7        : chr  "No" "No" "No" "No" ...
##  $ PctDiscMM     : num  0 0.151 0 0 0 ...
##  $ PctDiscCH     : num  0 0 0.0914 0 0 ...
##  $ ListPriceDiff : num  0.24 0.24 0.23 0 0 0.3 0.3 0.24 0.24 0.24 ...
##  $ STORE         : int  1 1 1 1 0 0 0 0 0 0 ...

head(orange[, 1:10])

##   Purchase WeekofPurchase StoreID PriceCH PriceMM DiscCH DiscMM SpecialCH
## 1       CH            237       1    1.75    1.99   0.00    0.0         0
## 2       CH            239       1    1.75    1.99   0.00    0.3         0
```

```
## 3        CH            245        1     1.86     2.09    0.17    0.0          0
## 4        MM            227        1     1.69     1.69    0.00    0.0          0
## 5        CH            228        7     1.69     1.69    0.00    0.0          0
## 6        CH            230        7     1.69     1.99    0.00    0.0          0
##    SpecialMM  LoyalCH
## 1          0 0.500000
## 2          1 0.600000
## 3          0 0.680000
## 4          0 0.400000
## 5          0 0.956535
## 6          1 0.965228
```

Now we will split the data into training and testng for this purpose, we will use createDataPartition method. set.seed(100) to have the random data be the same every time we run

```
# Create the training and test datasets
set.seed(100)

# Step 1: Get row numbers for the training data
trainRowNumbers <- createDataPartition(orange$Purchase, p=0.8, list=FALSE)

# Step 2: Create the training  dataset
trainData <- orange[trainRowNumbers,]

# Step 3: Create the test dataset
testData <- orange[-trainRowNumbers,]

# Store X and Y for later use.
x = trainData[, 2:18]
y = trainData$Purchase
```

Before we do further data processing on the data, we can also check some stats about the dataset. The skimer package provide a good solution to do so.

```
library(skimr)

## Warning: package 'skimr' was built under R version 4.2.3

skimmed <- skim_to_wide(trainData)

## Warning: 'skim_to_wide' is deprecated.
## Use 'skim()' instead.
## See help("Deprecated")

skimmed[, c(1:5, 9:11, 13, 15:16)]
```

Data summary

Name	Piped data
Number of rows	857

Number of columns	18
Column type frequency:	
character	2
numeric	16
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	whitespace
Purchase	0	1	2	0
Store7	0	1	2	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p25	p75	p100
WeekofPurchase	0	1.00	254.16	15.64	240.00	268.00	278.00
StoreID	1	1.00	4.01	2.33	2.00	7.00	7.00
PriceCH	0	1.00	1.87	0.10	1.79	1.99	2.09
PriceMM	2	1.00	2.08	0.14	1.99	2.18	2.29
DiscCH	1	1.00	0.05	0.12	0.00	0.00	0.50
DiscMM	4	1.00	0.13	0.22	0.00	0.24	0.80
SpecialCH	2	1.00	0.15	0.36	0.00	0.00	1.00
SpecialMM	5	0.99	0.17	0.37	0.00	0.00	1.00
LoyalCH	3	1.00	0.56	0.31	0.33	0.84	1.00
SalePriceMM	5	0.99	1.96	0.26	1.69	2.13	2.29
SalePriceCH	1	1.00	1.81	0.15	1.75	1.89	2.09
PriceDiff	0	1.00	0.14	0.27	0.00	0.32	0.64
PctDiscMM	4	1.00	0.06	0.10	0.00	0.12	0.40
PctDiscCH	2	1.00	0.03	0.06	0.00	0.00	0.25
ListPriceDiff	0	1.00	0.22	0.11	0.14	0.30	0.44
STORE	2	1.00	1.59	1.43	0.00	3.00	4.00

Now, we will fill the missing values with the TrainData dataset. The most common algorithm used for this purpose is KNN. We will use preprocess and predict function to do this task.

```r
# Create the knn imputation model on the training data
preProcess_missingdata_model <- preProcess(trainData, method='knnImpute')
preProcess_missingdata_model
```

```
## Created from 827 samples and 18 variables
##
## Pre-processing:
##   - centered (16)
##   - ignored (2)
##   - 5 nearest neighbor imputation (16)
##   - scaled (16)
```

```
# Use the imputation model to predict the values of missing data points
library(RANN)  # required for knnImpute
```

```
## Warning: package 'RANN' was built under R version 4.2.3
```

```
trainData <- predict(preProcess_missingdata_model, newdata = trainData)
anyNA(trainData)
```

```
## [1] FALSE
```

It is common to have categorical variables in the dataset. In order to convert to numerical to be useful in the machine learning models, we can implemnet the one-hot-encoding using the dummyVars() as function as the following:

```
dummies_model <- dummyVars(Purchase ~ ., data=trainData)

trainData_mat <- predict(dummies_model, newdata = trainData)

trainData <- data.frame(trainData_mat)

str(trainData)
```

```
## 'data.frame':    857 obs. of  18 variables:
##  $ WeekofPurchase: num  -1.097 -0.969 -0.586 -1.737 -1.673 ...
##  $ StoreID       : num  -1.29 -1.29 -1.29 -1.29 1.29 ...
##  $ PriceCH       : num  -1.1422 -1.1422 -0.0592 -1.7329 -1.7329 ...
##  $ PriceMM       : num  -0.6795 -0.6795 0.0498 -2.8676 -2.8676 ...
##  $ DiscCH        : num  -0.444 -0.444 0.981 -0.444 -0.444 ...
##  $ DiscMM        : num  -0.578 0.793 -0.578 -0.578 -0.578 ...
##  $ SpecialCH     : num  -0.425 -0.425 -0.425 -0.425 -0.425 ...
##  $ SpecialMM     : num  -0.447 2.235 -0.447 -0.447 -0.447 ...
##  $ LoyalCH       : num  -0.211 0.116 0.378 -0.539 1.284 ...
##  $ SalePriceMM   : num  0.13 -1.037 0.519 -1.037 -1.037 ...
##  $ SalePriceCH   : num  -0.432 -0.432 -0.843 -0.843 -0.843 ...
##  $ PriceDiff     : num  0.352 -0.744 0.936 -0.525 -0.525 ...
##  $ Store7No      : num  1 1 1 1 0 0 0 0 0 0 ...
##  $ Store7Yes     : num  0 0 0 0 1 1 1 1 1 1 ...
##  $ PctDiscMM     : num  -0.587 0.861 -0.587 -0.587 -0.587 ...
##  $ PctDiscCH     : num  -0.44 -0.44 1 -0.44 -0.44 ...
##  $ ListPriceDiff : num  0.21 0.21 0.118 -2.012 -2.012 ...
##  $ STORE         : num  -0.412 -0.412 -0.412 -0.412 -1.111 ...
```

```
preProcess_range_model <- preProcess(trainData, method = 'range')
trainData <- predict(preProcess_range_model, newdata = trainData)

trainData$Purchase <- y
```

We have many machine learning models supported by caret as shown below:

```
modelnames <- paste(names(getModelInfo()), collapse=',  ')
modelnames
```

```
## [1] "ada,  AdaBag,  AdaBoost.M1,  adaboost,  amdai,  ANFIS,  avNNet,
awnb,  awtan,  bag,  bagEarth,  bagEarthGCV,  bagFDA,  bagFDAGCV,  bam,
bartMachine,  bayesglm,  binda,  blackboost,  blasso,  blassoAveraged,
bridge,  brnn,  BstLm,  bstSm,  bstTree,  C5.0,  C5.0Cost,  C5.0Rules,
C5.0Tree,  cforest,  chaid,  CSimca,  ctree,  ctree2,  cubist,  dda,
deepboost,  DENFIS,  dnn,  dwdLinear,  dwdPoly,  dwdRadial,  earth,  elm,
enet,  evtree,  extraTrees,  fda,  FH.GBML,  FIR.DM,  foba,  FRBCS.CHI,
FRBCS.W,  FS.HGD,  gam,  gamboost,  gamLoess,  gamSpline,  gaussprLinear,
gaussprPoly,  gaussprRadial,  gbm_h2o,  gbm,  gcvEarth,  GFS.FR.MOGUL,
GFS.LT.RS,  GFS.THRIFT,  glm.nb,  glm,  glmboost,  glmnet_h2o,  glmnet,
glmStepAIC,  gpls,  hda,  hdda,  hdrda,  HYFIS,  icr,  J48,  JRip,
kernelpls,  kknn,  knn,  krlsPoly,  krlsRadial,  lars,  lars2,  lasso,  lda,
lda2,  leapBackward,  leapForward,  leapSeq,  Linda,  lm,  lmStepAIC,  LMT,
loclda,  logicBag,  LogitBoost,  logreg,  lssvmLinear,  lssvmPoly,
lssvmRadial,  lvq,  M5,  M5Rules,  manb,  mda,  Mlda,  mlp,  mlpKerasDecay,
mlpKerasDecayCost,  mlpKerasDropout,  mlpKerasDropoutCost,  mlpML,  mlpSGD,
mlpWeightDecay,  mlpWeightDecayML,  monmlp,  msaenet,  multinom,  mxnet,
mxnetAdam,  naive_bayes,  nb,  nbDiscrete,  nbSearch,  neuralnet,  nnet,
nnls,  nodeHarvest,  null,  OneR,  ordinalNet,  ordinalRF,  ORFlog,  ORFpls,
ORFridge,  ORFsvm,  ownn,  pam,  parRF,  PART,  partDSA,  pcaNNet,  pcr,
pda,  pda2,  penalized,  PenalizedLDA,  plr,  pls,  plsRglm,  polr,  ppr,
pre,  PRIM,  protoclass,  qda,  QdaCov,  qrf,  qrnn,  randomGLM,  ranger,
rbf,  rbfDDA,  Rborist,  rda,  regLogistic,  relaxo,  rf,  rFerns,  RFlda,
rfRules,  ridge,  rlda,  rlm,  rmda,  rocc,  rotationForest,
rotationForestCp,  rpart,  rpart1SE,  rpart2,  rpartCost,  rpartScore,
rqlasso,  rqnc,  RRF,  RRFglobal,  rrlda,  RSimca,  rvmLinear,  rvmPoly,
rvmRadial,  SBC,  sda,  sdwd,  simpls,  SLAVE,  slda,  smda,  snn,
sparseLDA,  spikeslab,  spls,  stepLDA,  stepQDA,  superpc,
svmBoundrangeString,  svmExpoString,  svmLinear,  svmLinear2,  svmLinear3,
svmLinearWeights,  svmLinearWeights2,  svmPoly,  svmRadial,  svmRadialCost,
svmRadialSigma,  svmRadialWeights,  svmSpectrumString,  tan,  tanSearch,
treebag,  vbmpRadial,  vglmAdjCat,  vglmContRatio,  vglmCumulative,
widekernelpls,  WM,  wsrf,  xgbDART,  xgbLinear,  xgbTree,  xyf"
```

In the following section we will train a random forset model

```
modelLookup('earth')
```

```
##   model parameter          label forReg forClass probModel
## 1 earth    nprune         #Terms   TRUE     TRUE      TRUE
## 2 earth    degree Product Degree   TRUE     TRUE      TRUE
```

```r
# Set the seed for reproducibility
set.seed(100)

model_mars = train(Purchase ~ ., data=trainData, method = 'earth')

## Loading required package: earth

## Warning: package 'earth' was built under R version 4.2.3

## Loading required package: Formula

## Loading required package: plotmo

## Warning: package 'plotmo' was built under R version 4.2.3

## Loading required package: plotrix

## Loading required package: TeachingDemos

## Warning: package 'TeachingDemos' was built under R version 4.2.3

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

fitted <- predict(model_mars)
model_mars

## Multivariate Adaptive Regression Spline
##
## 857 samples
##  18 predictor
##   2 classes: 'CH', 'MM'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 857, 857, 857, 857, 857, 857, ...
## Resampling results across tuning parameters:
##
##   nprune  Accuracy   Kappa
##    2      0.8116999  0.5969106
##    9      0.8234148  0.6245781
##   17      0.8105738  0.5975440
##
## Tuning parameter 'degree' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were nprune = 9 and degree = 1.
```

Now, we have the machine learning model (mars). We will tast this model using the test dataset that we kept earlier. First we will preprocess the test datasetas the folowing:

```r
# Step 1: Impute missing values
testData2 <- predict(preProcess_missingdata_model, testData)
```

```r
# Step 2: Create one-hot encodings (dummy variables)
testData3 <- predict(dummies_model, testData2)

# Step 3: Transform the features to range between 0 and 1
testData4 <- predict(preProcess_range_model, testData3)

# View
head(testData4[, 1:10])
```

```
##    WeekofPurchase   StoreID PriceCH   PriceMM DiscCH DiscMM SpecialCH
SpecialMM
## 7      0.09803922 1.0000000   0.000 0.5000000      0    0.5         1
1
## 11     0.25490196 1.0000000   0.425 0.6666667      0    0.0         0
0
## 18     0.80392157 0.1666667   0.425 0.8166667      0    0.0         0
1
## 21     0.58823529 1.0000000   0.425 0.8166667      0    0.0         0
0
## 33     0.94117647 0.1666667   0.675 0.8166667      0    1.0         0
1
## 35     0.47058824 0.3333333   0.750 0.9000000      0    0.0         0
0
##      LoyalCH SalePriceMM
## 7  0.9722332   0.3636364
## 11 0.9886583   0.8181818
## 18 0.4000146   0.9000000
## 21 0.6000274   0.9000000
## 33 0.6800325   0.1727273
## 35 0.5440238   0.9454545
```

Now we will use the trained model to analyze the tast data and provide us with prediction

```r
# Predict on testData
predicted <- predict(model_mars, testData4)
head(predicted)
```

```
## [1] CH CH CH CH MM CH
## Levels: CH MM
```

Now, we will predict compare the predicted values against the actual values.

```r
confusionMatrix(reference = as.factor(testData$Purchase), data = predicted,
mode = 'everything', positive = 'MM')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 114  26
##         MM  16  57
```

```
## 
##                Accuracy : 0.8028
##                  95% CI : (0.743, 0.854)
##     No Information Rate : 0.6103
##     P-Value [Acc > NIR] : 1.281e-09
## 
##                   Kappa : 0.5762
## 
##  Mcnemar's Test P-Value : 0.1649
## 
##             Sensitivity : 0.6867
##             Specificity : 0.8769
##          Pos Pred Value : 0.7808
##          Neg Pred Value : 0.8143
##               Precision : 0.7808
##                  Recall : 0.6867
##                      F1 : 0.7308
##              Prevalence : 0.3897
##          Detection Rate : 0.2676
##    Detection Prevalence : 0.3427
##       Balanced Accuracy : 0.7818
## 
##        'Positive' Class : MM
## 
```

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00
```

For many machine learning models we can implemnt some performance tuning for the model. The perfoemance tuning model aims to have a higher accuracy for the model. It is very common to do this for any model you create.

```
#performance tuning
fitControl <- trainControl(
```

```
  method = 'cv', #k - folds validation
  number = 5,
  savePredictions = 'final', #saves prediction for oprimal tuning parameters
  classProbs = T, #should pass probabilites to be returned
  summaryFunction = twoClassSummary #results
)

#step 1: Tune hyper parameters by setting tuneLength
set.seed(100)
model_mars2 = train(Purchase ~ ., data = trainData, methods = 'earth',
tuneLength = 5, metrics = 'ROC', trControl = fitControl)

## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy"
was not
## in the result set. ROC will be used instead.

model_mars2

## Random Forest
##
## 857 samples
##  18 predictor
##   2 classes: 'CH', 'MM'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 685, 686, 685, 686, 686
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##    2    0.8711563  0.8660989  0.6615106
##    6    0.8871323  0.8565751  0.7333333
##   10    0.8867648  0.8527656  0.7573496
##   14    0.8862704  0.8565751  0.7602895
##   18    0.8850728  0.8508608  0.7723202
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 6.

# #step2 : predict and testData and compute the confusion metrics
# predicted2 <- predict(model_mars2, testData4)
# confusionMatrix(reference = as.factor(testData$Purchase), data =
predicted2, mode = 'everything', positve = 'MM')
#
# #step3: Define the tuneGrid
# marsGrid <- expand.grid(nprune = c(2, 4, 6, 8, 10),
#                          degree = c(1,2,3))
# #model3
# set.seed(100)
# model_mars3 = train(Purchase~., data = trainData, methods = 'earth',
tuneGrid = marsGrid, metrics = 'ROC', trControl = fitControl)
```

```
# model_mars3
#
# predicted3 <- predict(model_mars3, testData4)
# confusionMatrix(reference = as.factor(testData$Purchase), data =
#predicted3, mode = 'everything', positve = 'MM')
```

Finally we will train two more modles and we compare the performance of each one of them. After creating the models, we can compare their performance using resample function where you list all the models that you created.

```
set.seed(100)

#random forest

model_rf = train(Purchase ~ ., data = trainData, method = 'rf', tuneLength =
5, trControl = fitControl)

## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy"
was not
## in the result set. ROC will be used instead.

model_rf

## Random Forest
##
## 857 samples
##  18 predictor
##   2 classes: 'CH', 'MM'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 685, 686, 685, 686, 686
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##    2    0.8711563  0.8660989  0.6615106
##    6    0.8871323  0.8565751  0.7333333
##   10    0.8867648  0.8527656  0.7573496
##   14    0.8862704  0.8565751  0.7602895
##   18    0.8850728  0.8508608  0.7723202
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 6.

set.seed(100)

#svm

model_svmRadial = train(Purchase ~ ., data = trainData, method = 'svmRadial',
tuneLength = 15, trControl = fitControl)
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy"
was not
## in the result set. ROC will be used instead.

model_svmRadial

## Support Vector Machines with Radial Basis Function Kernel
##
## 857 samples
##  18 predictor
##   2 classes: 'CH', 'MM'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 685, 686, 685, 686, 686
## Resampling results across tuning parameters:
##
##   C        ROC        Sens       Spec
##      0.25  0.8968213  0.8795055  0.7274084
##      0.50  0.8980530  0.8776007  0.7214835
##      1.00  0.8977832  0.8776190  0.7334238
##      2.00  0.8934719  0.8718681  0.7303483
##      4.00  0.8915500  0.8794689  0.7154229
##      8.00  0.8868855  0.8890293  0.6825418
##     16.00  0.8823947  0.8870696  0.6854817
##     32.00  0.8767745  0.8889744  0.6583899
##     64.00  0.8600145  0.8889744  0.6524197
##    128.00  0.8486717  0.8813370  0.6494346
##    256.00  0.8413847  0.8832784  0.6284487
##    512.00  0.8313846  0.8871062  0.6196744
##   1024.00  0.8198163  0.8909524  0.6136137
##   2048.00  0.8143498  0.8986081  0.5598372
##   4096.00  0.8113379  0.9024725  0.5388964
##
## Tuning parameter 'sigma' was held constant at a value of 0.06525857
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.06525857 and C = 0.5.

#Compare all 3 using resample

models_compare <- resamples(list(RF = model_rf, MARS = model_mars2, SVM =
model_svmRadial))

summary(models_compare)

##
## Call:
## summary.resamples(object = models_compare)
##
## Models: RF, MARS, SVM
## Number of resamples: 5
```

```
##
## ROC
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## RF   0.8691198 0.8697618 0.8932262 0.8871323 0.8997868 0.9037669    0
## MARS 0.8691198 0.8697618 0.8932262 0.8871323 0.8997868 0.9037669    0
## SVM  0.8712843 0.8823192 0.9022033 0.8980530 0.9166188 0.9178394    0
##
## Sens
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## RF   0.8076923 0.8380952 0.8666667 0.8565751 0.8761905 0.8942308    0
## MARS 0.8076923 0.8380952 0.8666667 0.8565751 0.8761905 0.8942308    0
## SVM  0.8173077 0.8666667 0.8857143 0.8776007 0.8952381 0.9230769    0
##
## Spec
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## RF   0.6666667 0.6716418 0.7462687 0.7333333 0.7761194 0.8059701    0
## MARS 0.6666667 0.6716418 0.7462687 0.7333333 0.7761194 0.8059701    0
## SVM  0.6716418 0.6969697 0.7164179 0.7214835 0.7313433 0.7910448    0
```