Crossvalidation for Machine Learning in R

Arpan

January 4, 2017

Crossvalidation is a technique which gives the insight on how the model will generalize to an unknown or unseen dataset(test data set) by reducing the problems like overfitting.

A model is usually given a known data set(training data set) on which training is done and unknown dataset(testing data set) against which the model is tested.

Holdout method : The data set is partitioned into two parts, one is called the training set and other is the testing set.Then the model predicts the target variable for the testing data.

K-fold cross validation : The data set is divided into k subsets. Each time, one of the k subsets is used as the test set and the other k-1 subsets alltogether forms our training set. Then the average error across all k trials is computed.That means in K-fold cross-validation model is fitted K times and also tested K-times against the left-out subset of data.

Leave-one-out cross validation :It's a K-fold cross validation where K is equal to the number of data points in the set(i.e number of rows).That implies the model will be fitted N number of times where N is equal to number of rows.So if the number of rows is very large then this method will run many times and so it is very computationally expensive.

Summary: 1.In holdout method: We test the model only one time and that's also against one same subset of whole data set.Ofcourse you can choose subset according to your choice but its best to choose randomly. 2.K-fold crossvalidation:In this model runs K times . a.If K=1 then that is same as holdout method. b.If K=N(number of rows in data) then that is same as Leave-one-out crossvalidaton. 3.Choosing the best number of folds depends on data size,keeping in mind about computational expenses,etc. 4. Lower K

```
a.computationally cheaper,
b.less error due to variance
c.more error due to bias(model mismatch).
```

Higher K

```
a.more expensive
b.more error due to variance
c.lower error due to bias(model mismatch).
```

How to reduce Variance without increasing bias? Repeat the cross-validation with the same K but different random folds and then averaging the results but cons is that this is even more expensive.

Now let's have a look on how to do crossvalidation in R using the package caret.

Setting the seed so that we get the same results each time we run the model

```
set.seed(123)
```

Importing the library MASS for iris dataset and library caret for crossvalidation

```
library(MASS,quietly = TRUE)
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.2.3
```

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

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

```
## Warning: package 'ggplot2' was built under R version 3.2.5
```

Storing the data set named “iris” into DataFrame named “DataFrame”

```
DataFrame <- iris
```

Type help(“iris”) to know about the data set

```
help("iris")
```

```
## starting httpd help server ...
```

```
##  done
```

Lets check out the structure of the data

```
str(DataFrame)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Check the dimension of this data frame

```
dim(DataFrame)
```

```
## [1] 150   5
```

Check first 3 rows

```
head(DataFrame,3)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
```

Check the summary of data

```
summary(DataFrame)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

Check the number of unique values

```
apply(DataFrame,2,function(x) length(unique(x)))
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
##           35           23           43           22            3
```

Lets check the data set again

```
str(DataFrame)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Lets create the train and test data set.Target variable is Species

```
library(caTools)
library(caret)
ind = createDataPartition(DataFrame$Species, p = 2/3, list = FALSE)
trainDF<-DataFrame[ind,]
testDF<-DataFrame[-ind,]
```

We will be using the caret package for crossvalidation.Function named train in caret package is used for crossvalidation. Let's choose the paramters for the train function in caret

```
ControlParamteres <- trainControl(method = "cv",
                                  number = 5,
                                  savePredictions = TRUE,
                                  classProbs = TRUE
)
```

Let's choose the model parameters.Here we are choosing mtry of Random forest and taking three values.You can choose other model also and its parameters in the function expand.grid which will create a grid of all combinations of parameters.

```
parameterGrid <- expand.grid(mtry=c(2,3,4))
```

method="cv" (used for crossvalidation) number=5 (means 5 fold crossvalidation) classProbs=TRUE(model will save the predictions for each class)

We will put the above paramter in the model below in trControl argument Let's now fit the model using train function To know more about the train function type and run ?train in the console

```
modelRandom <- train(Species~.,
                     data = trainDF,
                     method = "rf",
                     trControl = ControlParamteres,
                     preProcess = c('center', 'scale'),
                     tuneGrid=parameterGrid
)
```

```
## Loading required package: randomForest
```

```
## Warning: package 'randomForest' was built under R version 3.2.2
```

```
## randomForest 4.6-12
```

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

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

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

method="rf"(means random forest.I have chosen random forest.You choose any model name here) preProcess=used for centering and scaling of the data. There are many other options available for other needs.Just hit the tab button after the comma inside the train function to read about the options available.

To know which models(or methods) are available other than random forest .Just type and run.

```
names(getModelInfo())
```

```
##   [1] "ada"                "AdaBag"             "AdaBoost.M1"
##   [4] "amdai"              "ANFIS"              "avNNet"
##   [7] "awnb"               "awtan"              "bag"
##  [10] "bagEarth"           "bagEarthGCV"        "bagFDA"
##  [13] "bagFDAGCV"          "bartMachine"        "bayesglm"
##  [16] "bdk"                "binda"              "blackboost"
##  [19] "blasso"             "blassoAveraged"     "Boruta"
##  [22] "bridge"             "brnn"               "BstLm"
##  [25] "bstSm"              "bstTree"            "C5.0"
```

```
##  [28] "C5.0Cost"           "C5.0Rules"          "C5.0Tree"
##  [31] "cforest"            "chaid"              "CSimca"
##  [34] "ctree"              "ctree2"             "cubist"
##  [37] "DENFIS"             "dnn"                "dwdLinear"
##  [40] "dwdPoly"            "dwdRadial"          "earth"
##  [43] "elm"                "enet"               "enpls.fs"
##  [46] "enpls"              "evtree"             "extraTrees"
##  [49] "fda"                "FH.GBML"            "FIR.DM"
##  [52] "foba"               "FRBCS.CHI"          "FRBCS.W"
##  [55] "FS.HGD"             "gam"                "gamboost"
##  [58] "gamLoess"           "gamSpline"          "gaussprLinear"
##  [61] "gaussprPoly"        "gaussprRadial"      "gbm"
##  [64] "gcvEarth"           "GFS.FR.MOGUL"       "GFS.GCCL"
##  [67] "GFS.LT.RS"          "GFS.THRIFT"         "glm"
##  [70] "glmboost"           "glmnet"             "glmStepAIC"
##  [73] "gpls"               "hda"                "hdda"
##  [76] "HYFIS"              "icr"                "J48"
##  [79] "JRip"               "kernelpls"          "kknn"
##  [82] "knn"                "krlsPoly"           "krlsRadial"
##  [85] "lars"               "lars2"              "lasso"
##  [88] "lda"                "lda2"               "leapBackward"
##  [91] "leapForward"        "leapSeq"            "Linda"
##  [94] "lm"                 "lmStepAIC"          "LMT"
##  [97] "loclda"             "logicBag"           "LogitBoost"
## [100] "logreg"             "lssvmLinear"        "lssvmPoly"
## [103] "lssvmRadial"        "lvq"                "M5"
## [106] "M5Rules"            "manb"               "mda"
## [109] "Mlda"               "mlp"                "mlpML"
## [112] "mlpWeightDecay"     "mlpWeightDecayML"   "multinom"
## [115] "nb"                 "nbDiscrete"         "nbSearch"
## [118] "neuralnet"          "nnet"               "nnls"
## [121] "nodeHarvest"        "oblique.tree"       "OneR"
## [124] "ORFlog"             "ORFpls"             "ORFridge"
## [127] "ORFsvm"             "ownn"               "pam"
## [130] "parRF"              "PART"               "partDSA"
## [133] "pcaNNet"            "pcr"                "pda"
## [136] "pda2"               "penalized"          "PenalizedLDA"
## [139] "plr"                "pls"                "plsRglm"
## [142] "polr"               "ppr"                "protoclass"
## [145] "pythonKnnReg"       "qda"                "QdaCov"
## [148] "qrf"                "qrnn"               "randomGLM"
## [151] "ranger"             "rbf"                "rbfDDA"
## [154] "rda"                "relaxo"             "rf"
## [157] "rFerns"             "RFlda"              "rfRules"
## [160] "ridge"              "rknn"               "rknnBel"
## [163] "rlm"                "rmda"               "rocc"
## [166] "rotationForest"     "rotationForestCp"   "rpart"
## [169] "rpart1SE"           "rpart2"             "rpartCost"
## [172] "rqlasso"            "rqnc"               "RRF"
## [175] "RRFglobal"          "rrlda"              "RSimca"
## [178] "rvmLinear"          "rvmPoly"            "rvmRadial"
## [181] "SBC"                "sda"                "sddaLDA"
## [184] "sddaQDA"            "sdwd"               "simpls"
## [187] "SLAVE"              "slda"               "smda"
```

```
## [190] "snn"                 "sparseLDA"           "spikeslab"
## [193] "spls"                "stepLDA"             "stepQDA"
## [196] "superpc"             "svmBoundrangeString" "svmExpoString"
## [199] "svmLinear"           "svmLinear2"          "svmPoly"
## [202] "svmRadial"           "svmRadialCost"       "svmRadialSigma"
## [205] "svmRadialWeights"    "svmSpectrumString"   "tan"
## [208] "tanSearch"           "treebag"             "vbmpRadial"
## [211] "widekernelpls"       "WM"                  "wsrf"
## [214] "xgbLinear"           "xgbTree"             "xyf"
```

To know about the random forest model we just fitted,just type model name

```
modelRandom
```

```
## Random Forest
##
## 102 samples
##   4 predictor
##   3 classes: 'setosa', 'versicolor', 'virginica'
##
## Pre-processing: centered (4), scaled (4)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 81, 81, 82, 83, 81
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##   2     0.9609023  0.9412433  0.04020194   0.06034973
##   3     0.9809524  0.9714286  0.02608203   0.03912304
##   4     0.9809524  0.9714286  0.02608203   0.03912304
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 3.
```

The tuning parameter is mtry.The accuracy metric for each mtry value is given in the table .The best model is chosen with the mtry=2.

Let's check the predictions on the test data set

```
predictions<-predict(modelRandom,testDF)
```

Let's check the confusion matrix

```
t<-table(predictions=predictions,actual=testDF$Species)
t
```

```
##             actual
## predictions  setosa versicolor virginica
##   setosa         16          0         0
##   versicolor      0         14         3
##   virginica       0          2        13
```