Practical Data Science: Reducing High Dimensional Data in R

Manuel Amunategui - amunategui@gmail.com

Feature Selection using Ensembles and the {fscaret} package

The fscaret package, as its name implies, is closely related to the caret package. It relies on **caret**, and its numerous functions, to get its job done.

So what does this package do?
Well, you give it a data set and a list of models and, in return, **fscaret** will scale and return the importance of each variable for each model and for the ensemble of models. The tool extracts the importance of each variable by using the selected models' VarImp or similar measuring function. For example, linear models use the absolute value of the t-statistic for each parameter and decision-tree models, total the importance of the individual trees, etc.

We saw **caret** in the previous lectures, but if you're jumping in, it supports hundreds of **R** models and offers many machine learning and tuning functions. **fscaret** sits atop **caret** as an ensemble variable selection tool - variable selection on steroids!

To get a full list of all the models available with the **fscaret** package, install and load the fscaret package then type in the following:

```
library(fscaret)
```

See the models it supports in both 'regression' and 'classification' categories:

```
data(funcRegPred)
print(funcRegPred)
```

```
##   [1] "ANFIS"                 "avNNet"                "bag"
##   [4] "bagEarth"              "bayesglm"              "bdk"
##   [7] "blackboost"            "Boruta"                "bstLs"
##  [10] "bstSm"                 "bstTree"               "cforest"
##  [13] "ctree"                 "ctree2"                "cubist"
##  [16] "DENFIS"                "dnn"                   "earth"
##  [19] "elm"                   "enet"                  "evtree"
##  [22] "extraTrees"            "FIR.DM"                "foba"
##  [25] "FS.HGD"                "gam"                   "gamboost"
##  [28] "gamLoess"              "gamSpline"             "gaussprLinear"
##  [31] "gaussprPoly"           "gaussprRadial"         "gbm"
##  [34] "gcvEarth"              "GFS.FR.MOGAL"          "GFS.LT.RS"
##  [37] "GFS.Thrift"            "glm"                   "glmboost"
##  [40] "glmnet"                "glmStepAIC"            "HYFIS"
##  [43] "icr"                   "kernelpls"             "kknn"
##  [46] "knn"                   "krlsPoly"              "krlsRadial"
##  [49] "lars"                  "lars2"                 "lasso"
##  [52] "leapBackward"          "leapForward"           "leapSeq"
##  [55] "lm"                    "lmStepAIC"             "logicBag"
##  [58] "logreg"                "M5"                    "M5Rules"
##  [61] "mlp"                   "mlpWeightDecay"        "neuralnet"
##  [64] "nnet"                  "nodeHarvest"           "parRF"
##  [67] "partDSA"               "pcaNNet"               "pcr"
##  [70] "penalized"             "pls"                   "plsRglm"
##  [73] "ppr"                   "qrf"                   "qrnn"
##  [76] "relaxo"                "rf"                    "ridge"
##  [79] "rknn"                  "rknnBel"               "rlm"
##  [82] "rpart"                 "rpart2"                "RRF"
##  [85] "RRFglobal"             "rvmLinear"             "rvmPoly"
##  [88] "rvmRadial"             "SBC"                   "simpls"
##  [91] "spls"                  "superpc"
"svmBoundrangeString"
##  [94] "svmExpoString"         "svmLinear"             "svmPoly"
##  [97] "svmRadial"             "svmRadialCost"         "svmSpectrumString"
## [100] "treebag"               "widekernelpls"         "WM"
## [103] "xyf"
```

```
data(funcClassPred)
print(funcClassPred)
```

```
##   [1] "ada"                   "bagFDA"                "brnn"
##   [4] "C5.0"                  "C5.0Cost"              "C5.0Rules"
##   [7] "C5.0Tree"              "CSimca"                "fda"
##  [10] "FH.GBML"               "FRBCS.CHI"             "FRBCS.W"
##  [13] "GFS.GCCL"              "gpls"                  "hda"
##  [16] "hdda"                  "J48"                   "JRip"
##  [19] "lda"                   "lda2"                  "Linda"
##  [22] "LMT"                   "LogitBoost"            "lssvmLinear"
##  [25] "lssvmPoly"             "lssvmRadial"           "lvq"
##  [28] "mda"                   "Mlda"                  "multinom"
##  [31] "nb"                    "oblique.tree"          "OneR"
##  [34] "ORFlog"                "ORFpls"                "ORFridge"
##  [37] "ORFsvm"                "pam"                   "PART"
##  [40] "pda"                   "pda2"                  "PenalizedLDA"
##  [43] "plr"                   "protoclass"            "qda"
##  [46] "QdaCov"                "rbf"                   "rda"
##  [49] "rFerns"                "RFlda"                 "rocc"
##  [52] "rpartCost"             "rrlda"                 "RSimca"
##  [55] "sda"                   "sddaLDA"               "sddaQDA"
##  [58] "SLAVE"                 "slda"                  "smda"
##  [61] "sparseLDA"             "stepLDA"               "stepQDA"
##  [64] "svmRadialWeights"      "vbmpRadial"            "avNNet"
##  [67] "bag"                   "bagEarth"              "bayesglm"
##  [70] "bdk"                   "blackboost"            "Boruta"
##  [73] "bstLs"                 "bstSm"                 "bstTree"
##  [76] "cforest"               "ctree"                 "ctree2"
##  [79] "dnn"                   "earth"                 "elm"
##  [82] "evtree"                "extraTrees"            "gam"
##  [85] "gamboost"              "gamLoess"              "gamSpline"
##  [88] "gaussprLinear"         "gaussprPoly"           "gaussprRadial"
##  [91] "gbm"                   "gcvEarth"              "glm"
##  [94] "glmboost"              "glmnet"                "glmStepAIC"
##  [97] "kernelpls"             "kknn"                  "knn"
## [100] "logicBag"              "logreg"                "mlp"
## [103] "mlpWeightDecay"        "nnet"                  "nodeHarvest"
## [106] "parRF"                 "partDSA"               "pcaNNet"
## [109] "pls"                   "plsRglm"               "rf"
## [112] "rknn"                  "rknnBel"               "rpart"
## [115] "rpart2"                "RRF"                   "RRFglobal"
## [118] "simpls"                "spls"
"svmBoundrangeString"
## [121] "svmExpoString"         "svmLinear"             "svmPoly"
## [124] "svmRadial"             "svmRadialCost"         "svmSpectrumString"
## [127] "treebag"               "widekernelpls"         "xyf"
```

caret supports almost 200 models (this number keeps growing so check the latest numbers directly off of a fresh package - run length(getModelInfo()) after loading caret).

When installing **fscaret**, it is recommended to install it with all its dependencies:

```
# install.packages("fscaret", dependencies = c("Depends", "Suggests"))
```

This will speed things up tremendously on subsequent runs but you will have to suffer during the installation process. If I remember correctly, it took me over an hour to get all the models installed (some of the models require user confirmations). If you don't do this and attempt to run the fscaret function on a method not currently installed, the function will interrupt and it will prompt you to install it.

The input data needs to be formatted in a particular way: **MISO**. (Multiple Ins, Single Out). The output needs be the last column in the data frame. So you can't have it anywhere else, nor can it predict multiple response columns at once.

As with anything **ensemble** related, if you're going to run 50 models in one shot, you better have the computing muscle to do so - there's no free lunch. Start with a single or small set of models. If you're going to run a large ensemble of models, fire it up before going to bed and see what you get the next day.

For the lecture, we'll use a Titanic dataset from the University of Colorado Denver. This is a classic data set often seen in online courses and walkthroughs. The outcome is passenger survivorship (i.e. can you predict who will survive based on various features). We drop the passenger names as they are all unique but keep the passenger titles. We also impute the missing 'Age' variables with the mean:

```
titanicDF <-
read.csv('http://math.ucdenver.edu/RTutorial/titanic.txt',sep='\t')

titanicDF$Title <- ifelse(grepl('Mr
',titanicDF$Name),'Mr',ifelse(grepl('Mrs
',titanicDF$Name),'Mrs',ifelse(grepl('Miss',titanicDF$Name),'Miss','Nothing')))


titanicDF$Age[is.na(titanicDF$Age)] <- median(titanicDF$Age, na.rm=T)
```

We move the 'Survived' outcome variable to the end of the data frame to be MISO compliant:

```
# miso format
titanicDF <- titanicDF[c('PClass', 'Age',  'Sex',  'Title', 'Survived')]
```

To help us process this data, we're going to use some of caret functions. First, we call caret's dummyVars function to dummify the 'Title' variable which creates four new columns, one for each title type:

```
titanicDF$Title <- as.factor(titanicDF$Title)
titanicDummy <- dummyVars("~.",data=titanicDF, fullRank=F)
titanicDF <- as.data.frame(predict(titanicDummy,titanicDF))
print(names(titanicDF))
```

```
##  [1] "PClass.1st"    "PClass.2nd"    "PClass.3rd"    "Age"
##  [5] "Sex.female"    "Sex.male"      "Title.Miss"    "Title.Mr"
##  [9] "Title.Mrs"     "Title.Nothing" "Survived"
```

```
# to enable probabilities we need to force outcome to factor
titanicDF$Survived <- as.factor(ifelse(titanicDF$Survived==1, 'yes',
'no'))
```

We split the data set randomly using a 80% split. With this split we allocate 0.8 of the data to the training set and 0.2 to the testing set:

```
set.seed(1234)
splitIndex <- sample(nrow(titanicDF), floor(0.8*nrow(titanicDF)))
traindf <- titanicDF[ splitIndex,]
testdf  <- titanicDF[-splitIndex,]
```

Finally, we select three models to process our data and call the meat-and-potatoes function of the **fscaret** package, named as its package, **fscaret**:

```
myFS.class <-fscaret(traindf, testdf, myTimeLimit = 20,
                     preprocessData=TRUE, with.labels=TRUE,
classPred=TRUE,
                     regPred=FALSE,
Used.funcClassPred=c("gbm","rpart","pls"),
                     supress.output=FALSE, no.cores=NULL,
saveModel=FALSE)
```

```
## ----Processing files:----
## [1] "9in_default_CLASSControl_gbm.RData"
## [2] "9in_default_CLASSControl_pls.RData"
## [3] "9in_default_CLASSControl_rpart.RData"
## [1] ""
## [1] "Calculating error measure for model:"
## [1] "9in_default_CLASSControl_gbm.RData"
## [1] ""
```

If the above code ran successfully, you will see a series of log outputs (unless you set **supress.output** to false). Each model will run through its paces and the final **fscaret** output will list the number of variables each model processed:

```
----Processing files:----
[1] "9in_default_CLASSControl_VarImp_gbm.txt"
"9in_default_CLASSControl_VarImp_pls.txt"
"9in_default_CLASSControl_VarImp_rpart.txt"
```

Also, not all classification models will work for a particular data set. Play around with different models to find a good mix.

The **myFS** variable holds a lot of information. The one we're interested in is the **`$VarImp$matrixVarImp.MeasureError`**. This returns the top variables, error measurements from the perspective of all models involved, and a scaled scoring to compare models:

```
results <-  myFS.class$VarImp$matrixVarImp.MeasureError
```

We need to do a little wrangling in order to clean this up and get a nicely ordered list with the actual variable names attached:

```
results$Input_no <- as.numeric(results$Input_no)
results <- results[,setdiff(names(results), c('SUM%','ImpGrad'))]
myFS.class$PPlabels$Input_no <-
as.numeric(rownames(myFS.class$PPlabels))
results <- merge(x=results, y=myFS.class$PPlabels, by="Input_no",
all.x=T)
results <- results[order(-results$SUM),]
print(head(results))
```

```
##   Input_no    gbm   pls  rpart   SUM Orig Input No     Labels
## 3        3 31.069 17.27 24.320 72.66             3 PClass.3rd
## 7        7 25.263 22.20 21.214 68.68             8   Title.Mr
## 5        5 23.427 21.78 21.618 66.83             5 Sex.female
## 1        1  5.083 12.62 14.266 31.97             1 PClass.1st
## 8        8  4.610 12.65 12.743 30.01             9  Title.Mrs
## 4        4  9.575  0.00  1.227 10.80             4        Age
```

We see that **PClass.3rd, Title.Mr, and Sex.female** are favorites of all three models in aggregate.

So, as a fun project, let's run the top 5 variables through the **Partial Least Squares (PLS)** method and predict the score. Instead of using predictions in the form of **type='raw'** (which returns actual values), we'll use **type='prob'** (which reruns probabilities between 0 and 1). This will allow us to use the Area Under the Curve (AUC) score from the {pROC} package (install it if needed).

We'll run the models with the top **5** voted variables:

```
traindf_truncated <- traindf[, c(head(as.character(results$Labels),5),
'Survived')]
dim(traindf_truncated)
```

```
## [1] 1050    6
```

```
objControl <- trainControl(method='cv', number=3, returnResamp='none',
summaryFunction = twoClassSummary, classProbs = TRUE)

# pls model
set.seed(1234)
pls_model <- train(Survived~., data=traindf_truncated, method="pls",
metric='roc', trControl=objControl)
pls_predictions <- predict(object=pls_model,
testdf[,setdiff(names(traindf_truncated), 'Survived')], type='prob')
library(pROC)
print(auc(predictor=pls_predictions[[2]],response=
ifelse(testdf$Survived=='yes',1,0)))
```

```
## Area under the curve: 0.844
```

Let's have some fun and show you the basis of creating a loop of models (keep in mind that this requires a lot of experimentation as some models require special parameter settings - keep experimenting). Let's run our top 5 variables through the models **C5.0, gbm, and rf**:

```
method_names <- c("C5.0", "gbm", "rf")
for (method_name in method_names) {
        print(method_name)
        set.seed(1234)
        model <- train(Survived~., data=traindf_truncated,
method=funcClassPred[funcClassPred==method_name],
                metric='roc', trControl=objControl)
        predictions <- predict(object=model,
testdf[,setdiff(names(traindf_truncated), 'Survived')], type='prob')
        print(auc(predictor=predictions[[2]],response=
ifelse(testdf$Survived=='yes',1,0)))
}
```

```
## [1] "C5.0"
## Area under the curve: 0.844
## [1] "gbm"
## Area under the curve: 0.844
## [1] "rf"
## Area under the curve: 0.803
```