Permutation Importance Example

Pablo Portillo Garrigues

May 3, 2019

Contents

Introduction

One of the biggest challenge in Data Science is finding the appropiate variables to predict the outcome of interest. In this paper we will propose an algorithm based on the paper that won the Heritage Provider Network Health Prize.

Algorithm

Basically the algorithm permutes the values of the variables to see if whether the error increase or decrease.

The algorithm has the following procedure:

1. Computes the importance of each variable through the importance function of the XGBoost algorithm
2. Sorts the variables in decreasing order of importance
3. Computes the error on the different validation sets with the variables intact.
4. Computes the error on each validation set having permuted the variable corresponding to the order suggested by step 2
5. The user has analyze whether or not to keep a variable accordingly to the results

```
importPerm<-function(datos,costF,cv=3,rep=3,perm=3,control=2,seed,importancia=0){

  set.seed(456)
  datos<-datos[sample(nrow(datos)),]
  D<-lapply(1:control,function(x){
    fo<-floor((nrow(datos)/control)*(x-1))+1
    ff<-floor((nrow(datos)/control)*(x))
    if(x==control){
      ff<-nrow(datos)
    }
    datos[fo:ff,]
  })
  import<-data.frame()
  BA<-lapply(1:control,function(x){
    aux<-costF(D[[x]],cv,rep,seed)
  })
  vars<-0
  if(class(importancia)=="function"){
```

```
  import<-importancia(datos,seed = seed)
  import<-as.data.frame(import)
  print(import)
  vars<-import[order(import$Gain),]$Feature
  aux<-c()
  for(i in vars){
    aux<-c(aux,which(colnames(datos)==i))
  }
  vars<-aux

  }
  else{
    vars<-which(!(colnames(datos) %in% c("target","id")))
  }
  for(i in vars){
    res<-lapply(1:control,function(x){
      data<-D[[x]]
      aux<-lapply(1:perm,function(y){
        data[,i]<-data[sample(nrow(data)),i]
        costF(data,cv,rep,seed)
        })
      ctrl<-mean(unlist(aux))
      ctrl
    })
    cat(paste("La variable",colnames(datos)[i],"tiene una diferencia de error respecto a la base de:" ,

  }

}
```

The algorithm has the following inputs:

- **datos** is the data in which we are interested
- **cost** is the objective function defined by the user
- **cv** is the number of cross-validation to perform in the objective function
- **rep** is the number of times to repeat the cross-validation in the objective function
- **perm** is the number of times to permute each variable (the higher the more robust the results)
- **control** is the number of subsets we are going to generate to validate the resuls (the higher the more robust the results)
- **seed** is the seed the algorithm is going to use
- **importancia** informs the algorithm which function to use in order to calculate the importance of each variable. If none specified (0) then the order in which the algorithm will calculate the permute importance will be the definied by the user

Example

Generate the data

Lets generate a synthetic example consisting on 13 variables, where the outcome is a non-linear relation between some variables.

```r
set.seed(240)
X<-as.data.frame(matrix(rnorm(130000),ncol=13))
colnames(X)
```

```
##  [1] "V1"  "V2"  "V3"  "V4"  "V5"  "V6"  "V7"  "V8"  "V9"  "V10" "V11"
## [12] "V12" "V13"
```

```r
#y<-X$V1*100*X$V2*30+5*X$V3*X$V4-X$V5^3-0.000001*X$V6^2+rnorm(nrow(X))
y<-20*X$V1*X$V2+4*X$V3*X$V4-X$V5^(3)-0.1*X$V6^(2)+rnorm(nrow(X))
data<-X
data$target<-y
data$id<-rnorm(nrow(data))
```

Cost function

Now we need to define a cost function. We will use the XGBoost algorithm to compute the error

```r
costPermXGB<-function(datos,cv,rep,seed){
  set.seed(seed)
  suppressMessages(library(dplyr))
  suppressMessages(library(MLmetrics))
  suppressMessages(library(xgboost))
  resRep<-c()
  for(j in 1:rep){
      cvId<-sample(1:cv,nrow(datos),replace=TRUE)
      fitness<-c()
      for(i in 1:cv){
        train<-datos[cvId!=i,]
        test<-datos[cvId==i,]
        train.y<-train$target
        test.y<-test$target
        train<-train[,!(colnames(train) %in% c("target","id"))]
        test<-test[,!(colnames(test) %in% c("target","id"))]
        new_tr <- model.matrix(~.+0,data = train)
    new_ts <- model.matrix(~.+0,data = test)
    dtrain <- xgb.DMatrix(data = new_tr,label = train.y)
    dtest <- xgb.DMatrix(data = new_ts,label = sample(y,nrow(test),replace = TRUE))
    xgb1 <- xgb.train(eval_metric="mae", data = dtrain, nrounds = 350, print_every_n = 10)
    preds<-predict(xgb1,dtest)
        fitness<-c(fitness,MAE(preds,test.y))
      }
      resRep<-c(resRep,mean(fitness))
    }
    mean(resRep)
}
```

Importance Function

Now we are going the define the importance function we are going to use. In this case we are going to use the XGB one, but we could also use the Random Forest one for example.

```r
importXGB<-function(datos,seed){
  set.seed(seed)
  suppressMessages(library(dplyr))
  suppressMessages(library(xgboost))

        train.y<-datos$target
        train<-datos[,!(colnames(datos) %in% c("target","id"))]
        new_tr <- model.matrix(~.+0,data = train)
    dtrain <- xgb.DMatrix(data = new_tr,label = train.y)
    xgb1 <- xgb.train(eval_metric="mae", data = dtrain, nrounds = 350, print_every_n = 10)
    return(xgb.importance(model=xgb1))
}
```

Permutation importance

In this data the only variables partitipating in the outcome are the first six, lets see if the algorithm detects it.

```r
importPerm(data,costF=costPermXGB,3,3,5,control = 2,seed=360,importancia = importXGB)
```

```
##    Feature        Gain      Cover  Frequency
## 1       V1 0.563875815 0.13888125 0.18425920
## 2       V2 0.354198715 0.13719797 0.13685869
## 3       V5 0.039895162 0.12658979 0.08774574
## 4       V3 0.017019635 0.07496037 0.08760874
## 5       V4 0.014860127 0.08073825 0.08048496
## 6       V9 0.001856386 0.06156392 0.05589424
## 7       V6 0.001661990 0.06764214 0.05774368
## 8      V12 0.001386686 0.04608175 0.05226385
## 9      V10 0.001135496 0.06755572 0.05466128
## 10      V7 0.001119993 0.05564860 0.05356531
## 11      V8 0.001099595 0.04454040 0.04972943
## 12     V11 0.001057150 0.04375377 0.05048291
## 13     V13 0.000833250 0.05484605 0.04870197
## La variable V13 tiene una diferencia de error respecto a la base de: -0.00927505085588403
## La variable V11 tiene una diferencia de error respecto a la base de: -0.00301719826872668
## La variable V8 tiene una diferencia de error respecto a la base de: -0.00916144784718638
## La variable V7 tiene una diferencia de error respecto a la base de: -0.0270745231362298
## La variable V10 tiene una diferencia de error respecto a la base de: -0.0316406014003187
## La variable V12 tiene una diferencia de error respecto a la base de: -0.0087954365801878
## La variable V6 tiene una diferencia de error respecto a la base de: -0.0218801960885428
## La variable V9 tiene una diferencia de error respecto a la base de: -0.0253576187651394
## La variable V4 tiene una diferencia de error respecto a la base de: 0.330861332746931
## La variable V3 tiene una diferencia de error respecto a la base de: 0.349558328400799
## La variable V5 tiene una diferencia de error respecto a la base de: 0.205089675853096
## La variable V2 tiene una diferencia de error respecto a la base de: 3.03878703142848
## La variable V1 tiene una diferencia de error respecto a la base de: 3.06753461065186
```

As we can see we would have detected all the variables that are present in the model except V6 due to its low value relative to the targets.