MLPS5

Hao Ran Li, Feiwen Liang, Leila Lan, Susu Zhu, Yitao Hu

01/05/2020

```
#import data and libraries
library(randomForest)
library(glmnet)
library(xgboost)
library(ggplot2)
library(reshape2)
library(data.table)
library(knitr)
```

Question1

We first created the data by simulation, then fit the models and compute out-of-sample MSE

```
#define a function to compute MSE
compute_MSE=function(Model,Testset,XGB=F){
  if(XGB==F)
    {
  fitted_vals=predict(Model,Testset[,-1])
  }
  else{
    fitted_vals=predict(Model,as.matrix(x=Testset[,-1]))
  }

  MSE=mean((fitted_vals-Testset[,1])^2)
  return(MSE)
}
#fix XGB hypter_parameters
XGB_params=list(booster='gbtree',
                objective='reg:linear',
                eta=0.3,
                gamma=0,
                max_depth=6)

MSE=data.frame(matrix(NA,ncol = 6,nrow = 500))
colnames(MSE)=c('OLSa','OLSb','RForesta','RForestb','XGBa','XGBb')

for (i in 1:500)
  {
#simulate the data
x1=rnorm(1500)
x2=rnorm(1500)
ya=1.5*x1-2*x2+rnorm(1500)
yb=rep(0,1500)
```

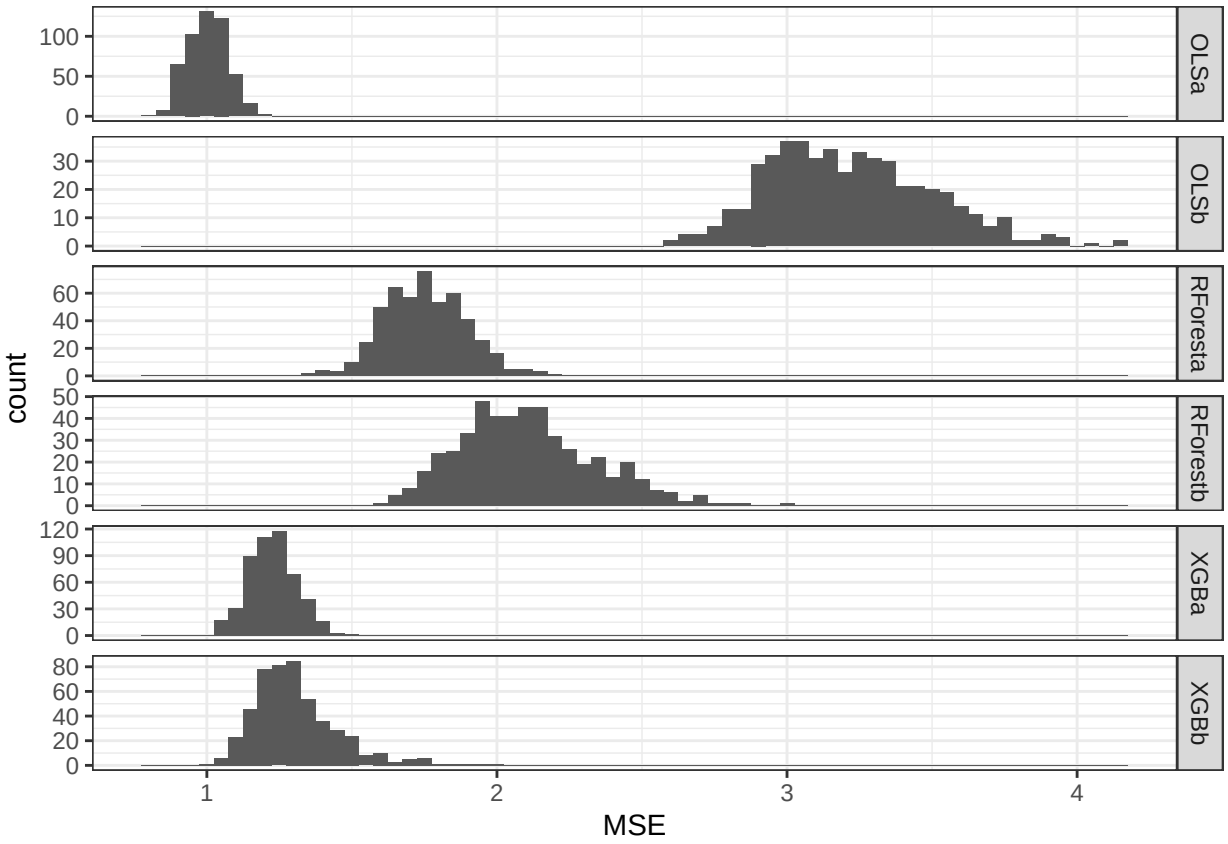
```
yb[which(x1>=0)]=1.5*log(x1[x1>=0])+rnorm(length(x1[x1>=0]))
yb[which(x1<0)]=1.5*x1[x1<0]-2*x2[which(x1<0)]+rnorm(length(x1[x1<0]))
Dataa=data.frame(cbind(ya,x1,x2))
Datab=data.frame(cbind(yb,x1,x2))
#split data into training and test set
Traina=Dataa[1:1000,]
Testa=Dataa[1001:1500,]
Trainb=Datab[1:1000,]
Testb=Datab[1001:1500,]
#fit the Models and compute out-of-sample MSE
OLSa=lm(formula = ya~x1+x2,data = Traina)
OLSb=lm(formula = yb~x1+x2,data = Trainb)

#Here, we only have 2 predictors and thus reduce the number of variable in each Bag fitting to 1 to red
RFa=randomForest(x = Traina[,-1],y=Traina[,1],ntree = 250,maxnodes = 10,mtry = 1)
RFb=randomForest(x = Trainb[,-1],y=Trainb[,1],ntree = 250,maxnodes = 10,mtry = 1)
#define the XGB data loader
XGBa_Train=xgb.DMatrix(data = as.matrix(x=Traina[,-1]),label=as.matrix(x=Traina[,1]))
XGBb_Train=xgb.DMatrix(data = as.matrix(Trainb[,-1]),label=as.matrix(Trainb[,1]))
#fit the XGB models
XGBa=xgb.cv(params = XGB_params,data = XGBa_Train,nrounds = 20,nfold = 10,verbose = F)
XGBb=xgb.cv(params = XGB_params,data = XGBb_Train,nrounds = 20,nfold = 10,verbose = F)
#get the best nrounds and refit the model
XGBopta=xgboost(params = XGB_params,
                data=XGBa_Train,
                nrounds = which.min(XGBa$evaluation_log$test_rmse_mean),
                verbose = F)
XGBoptb=xgboost(params = XGB_params,
                data=XGBb_Train,
                nrounds = which.min(XGBb$evaluation_log$test_rmse_mean),
                verbose = F)
#compute MSEs
MSE$OLSa[i]=compute_MSE(Model = OLSa,Testset = Testa)
MSE$OLSb[i]=compute_MSE(Model = OLSb,Testset = Testb)
MSE$RForesta[i]=compute_MSE(Model = RFa,Testset = Testa)
MSE$RForestb[i]=compute_MSE(Model = RFb,Testset = Testb)
MSE$XGBa[i]=compute_MSE(Model = XGBopta,Testset = Testa,XGB = T)
MSE$XGBb[i]=compute_MSE(Model = XGBoptb,Testset = Testb,XGB = T)
}
```

Plot the MSE histograms for all the models:

```
MSE=data.table(MSE)
MSE=melt(MSE,id.vars = NULL,measure.vars = NULL)
```

```
## Warning in melt.data.table(MSE, id.vars = NULL, measure.vars = NULL): id.vars
## and measure.vars are internally guessed when both are 'NULL'. All non-numeric/
## integer/logical type columns are considered id.vars, which in this case are
## columns []. Consider providing at least one of 'id' or 'measure' vars in future.
```

```
ggplot(data = MSE,aes(x=value))+geom_histogram(binwidth = 0.05)+
  facet_grid(variable~.,scales = 'free')+xlab('MSE')+theme_bw()
```



From the graphs above, we can see that OLS does best for data process a, while worst for data process b. This is because the imposed linear structure works well when it is consistent with the true data generating process, as in the case of a, but works poorly if the true data generating process is non-linear. In essence, this OLS linear model incurs large bias when used to fit nonlinear data.

Also, we observe that XGB models outperform Random Forest models in both a and b data generating process,while both models perform cross-validation to avoid overfitting. I would argue this is because we over-regularize the Random Forest model when setting mtry=1 (each tree is fit only on one feature). Because we reduced the data dimensions of each tree in Random Forest, it tends to incur greater bias as a much simplier model and thus underperform when the true data-generating process is two dimensional.

Question 2

```
#import data
library(readr)
housing_data= read_csv("housing.csv")
housing_data=as.data.frame(housing_data)
#training and test set split
Xtrain=housing_data[1:400,1:11]
ytrain=housing_data[1:400,12]
Xtest=housing_data[400:506,1:11]
ytest=housing_data[400:506,12]
```

a Random Forest

```
#fit the Model, setting mtry as sqrt of feature number
RFmodel=randomForest(x = Xtrain,y = ytrain,ntree = 500,maxnodes = 10,mtry = round(sqrt(11)))
```

```
#define a function to get the performance of this model
getPerformance=function(Model,Xtest,ytest,XGB=F){
  if(XGB==F)
    {
  fitted_vals=predict(Model,Xtest)
  }
  else{
    fitted_vals=predict(Model,as.matrix(x=Xtest))
  }
  MSE=mean((fitted_vals-ytest)^2)
  BenchmarkMSE=mean((mean(ytest)-ytest)^2)
  PsedoRsq=1-MSE/BenchmarkMSE
  out=c(MSE,BenchmarkMSE,PsedoRsq)
  names(out)=c('MSE','BenchmarkMSE','PsedoRsq')
  return(out)
}
out=getPerformance(Model = RFmodel,Xtest = Xtest,ytest = ytest)
kable(out)
```

	x
MSE	22.8285040
BenchmarkMSE	28.8172487
PsedoRsq	0.2078181

b XGBoost

```
#fix XGB hypter_parameters
XGB_params=list(booster='gbtree',
                objective='reg:linear',
                eta=0.1,
                gamma=0,
                max_depth=6)
#define the XGB data loader
XGB_Train=xgb.DMatrix(data = as.matrix(x=Xtrain),label=as.matrix(x=ytrain))
#cross-validate XGBoost model for the optimal nrounds
XGBcv=xgb.cv(params = XGB_params,data = XGB_Train,nrounds = 200,nfold = 10,verbose = F)
#get the best nrounds and refit the model
XGBopt=xgboost(params = XGB_params,
                data=XGB_Train,
                nrounds = which.min(XGBcv$evaluation_log$test_rmse_mean),verbose = F)
out=getPerformance(Model = XGBopt,Xtest = Xtest,ytest = ytest,XGB = T)
kable(out)
```

	x
MSE	20.0967201
BenchmarkMSE	28.8172487
PsedoRsq	0.3026149

c Linear Model with ElasticNet Regularization

```
#perform cross-validation for the linear model to get the optimal lambda
CVobj=cv.glmnet(x = as.matrix(Xtrain),y = as.vector(ytrain),alpha=0.5,type.measure = 'mse',nfolds = 4)
Opt.lambda=CVobj$lambda.min
#refit the model
GLMNETModel=glmnet(x = as.matrix(Xtrain),y = as.vector(ytrain),family = 'gaussian',alpha = 0.5,lambda =
#get the performance
out=getPerformance(Model = GLMNETModel,Xtest = Xtest,ytest = ytest,XGB = T)
kable(out)
```

	x
MSE	34.8588568
BenchmarkMSE	28.8172487
PsedoRsq	-0.2096525

The output above indicates that the regularized linear model underperforms the decision tree models (both Random Forest and XGBoost). It performs even worse than a model that guess the housing price at its mean. I would argue this is because the linear model is oversimplified, while the underground true relationship between housing price and the 11 features are nonlinear, which is better captured by a regularized decision tree model.

d Performance of Models with log transformantion of features

```
#log transform the specified features
housing_data[,c(3,5,8,10,11)]=log(housing_data[,c(3,5,8,10,11)])
Xtrain=housing_data[1:400,1:11]
ytrain=housing_data[1:400,12]
Xtest=housing_data[400:506,1:11]
ytest=housing_data[400:506,12]
#refit the models
RFmodel=randomForest(x = Xtrain,y = ytrain,ntree = 500,maxnodes = 10,mtry = round(sqrt(11)))
RFout=getPerformance(Model = RFmodel,Xtest = Xtest,ytest = ytest)
#define the XGB data loader
XGB_Train=xgb.DMatrix(data = as.matrix(x=Xtrain),label=as.matrix(x=ytrain))
#cross-validate XGBoost model for the optimal nrounds
XGBcv=xgb.cv(params = XGB_params,data = XGB_Train,nrounds = 200,nfold = 10,verbose = F)
#get the best nrounds and refit the model
XGBopt=xgboost(params = XGB_params,
               data=XGB_Train,
               nrounds = which.min(XGBcv$evaluation_log$test_rmse_mean),verbose = F)
XGBout=getPerformance(Model = XGBopt,Xtest = Xtest,ytest = ytest,XGB = T)
#perform cross-validation for the linear model to get the optimal lambda
CVobj=cv.glmnet(x = as.matrix(Xtrain),y = as.vector(ytrain),alpha=0.5,type.measure = 'mse',nfolds = 4)
Opt.lambda=CVobj$lambda.min
#refit the model
GLMNETModel=glmnet(x = as.matrix(Xtrain),y = as.vector(ytrain),family = 'gaussian',alpha = 0.5,lambda =
#get the performance
GLMNetout=getPerformance(Model = GLMNETModel,Xtest = Xtest,ytest = ytest,XGB = T)
kable(cbind(RFout,XGBout,GLMNetout))
```

	RFout	XGBout	GLMNetout
MSE	23.1075556	20.0967201	18.3407858
BenchmarkMSE	28.8172487	28.8172487	28.8172487
PsedoRsq	0.1981346	0.3026149	0.3635483

With log transformations, surprisingly, the linear model regurlarized by Elastic Net outperforms the decision trees. This is because the housing price probably depends on the growth level of the specified features instead of the absolute level themselves. The results indicate ,a limitation of regularization,that a properly specified parsimonious model (from economic theories or intuition) can actually outperform a regularized complex model, and therefore, regularization and cross-validation should not be misused only for the purpose of finding the appropriate model structure without having any sensible understanding between the features and labels.