STAT230 HW 11
University of California, Berkeley

Thibault Doutre, Student ID 26980469

April 28, 2016

1 Orthonormal variables

I first generate the data with standard independent normal variables.

```
makedata=function(p=20,n=100){
  X=matrix(rnorm(n*p),n,p)
  exps=seq(-1,-2.5,length=p)
  beta=rep(0,p)
  beta=exp(exps)
  Y=.5+X%*%beta+rnorm(n)
  data = data.frame(Y,X)
  colnames(data)=c("Y",letters[1:20])
  switch=sample(20)+1
  data=data[,c(1,switch)]
  names(beta)=names(data)[switch]
  return(data)
}
```

Then, I set a bunch of functions. Each of them computes a tenfolded cross validation of the method they use. The output is binary: y corresponds to the MSE, averaged over the 10 folds and x corresponds to the degree of freedom, which depends on the method used.

- $cv_o ls$ performs backward selection based on the t-value, with fitting an OLS method every time. The degrees of freedom are defined as the number of variables taken into the model.
- $cv_p cr$ performs PCR. The degrees of freedom are again defined as the number of variables taken into the model.

- cv_tree performs tree pruning after having fitted a tree model to the data. The degrees of freedom is defined as the size of the tree, i.e. the number of terminal nodes.
- cv_ridge performs ridge regression. The degrees of freedom are defined as $-\log\lambda$ where λ is the shrinkage coefficient. I then map this value to the 0-20 range in order to compare this method with the others.
- cv_lasso performs lasso regression. The degrees of freedom are defined as the number of non zero weights in front of the coefficients.

```
## # PCR
library(pls)

##
## Attaching package:  'pls'
## The following object is masked from 'package:stats':
##
##    loadings

cv_pcr = function(data){
  p = ncol(data)-1
  pcr.fit = pcr(Y~0+., ncomp=p, data=data, validation ="CV")
  cv_error = as.data.frame(RMSEP(pcr.fit)$val)[1,]
  names(cv_error) = 0:p
  return(list(y=cv_error,x=rev(0:p)))
}

## # OLS
cv_ols_one = function(data, nfold=10, formula="Y~."){
  Y_cv = c()
  nrows = nrow(data)
  MSE_test = c()
  for (i in seq(0, nrows-nfold, by=nrows/nfold)){
    test = (i+1):(nfold+i)
    train = -test
    lm.fit = lm(formula, data=data[train,])
    Ytest = predict(lm.fit, data[test,])
    MSE_test = c(MSE_test, mean((data$Y[test]-Ytest)^2))
  }
```

```
  return(mean(MSE_test))
}
cv_ols = function(data){
  p=ncol(data)-1
  nfold=10
  formula = "Y~."
  lm.fit = lm(formula, data=data)
  MSE_test = c()
  next_to_remove = ""
  variables = c()
  n = nrow(data)
  while(length(names(lm.fit$model))>1){
    MSE = cv_ols_one(data, nfold=nfold, formula)
    MSE_test = c(MSE,MSE_test)
    t_values = coef(summary(lm.fit))[,"t value"]
    next_to_remove = names(which.min(t_values[-1]))
    variables = c(next_to_remove,variables)
    formula = paste(formula,"-",next_to_remove,sep="")
    lm.fit = update(lm.fit, formula)
  }
  MSE = cv_ols_one(data, nfold, formula)
  MSE_test = c(MSE,MSE_test)
  return(list(y=MSE_test,x=0:p))
}


## # Tree
library(tree)
cv_tree <- function(a){
  assign("a", a, .GlobalEnv)
  tree.out=tree(Y~1+., data=a)
  cv = cv.tree(tree.out)
  MSE = cv$dev/100
  size = 20-cv$size
  y=rep(NaN,21)
  y[size]=MSE
  return(list(y=y,x=0:20))
}
```

```r
## # Ridge
library(glmnet)
```

```
## Loading required package:  Matrix
## Loading required package:  foreach
## Loaded glmnet 2.0-2
```

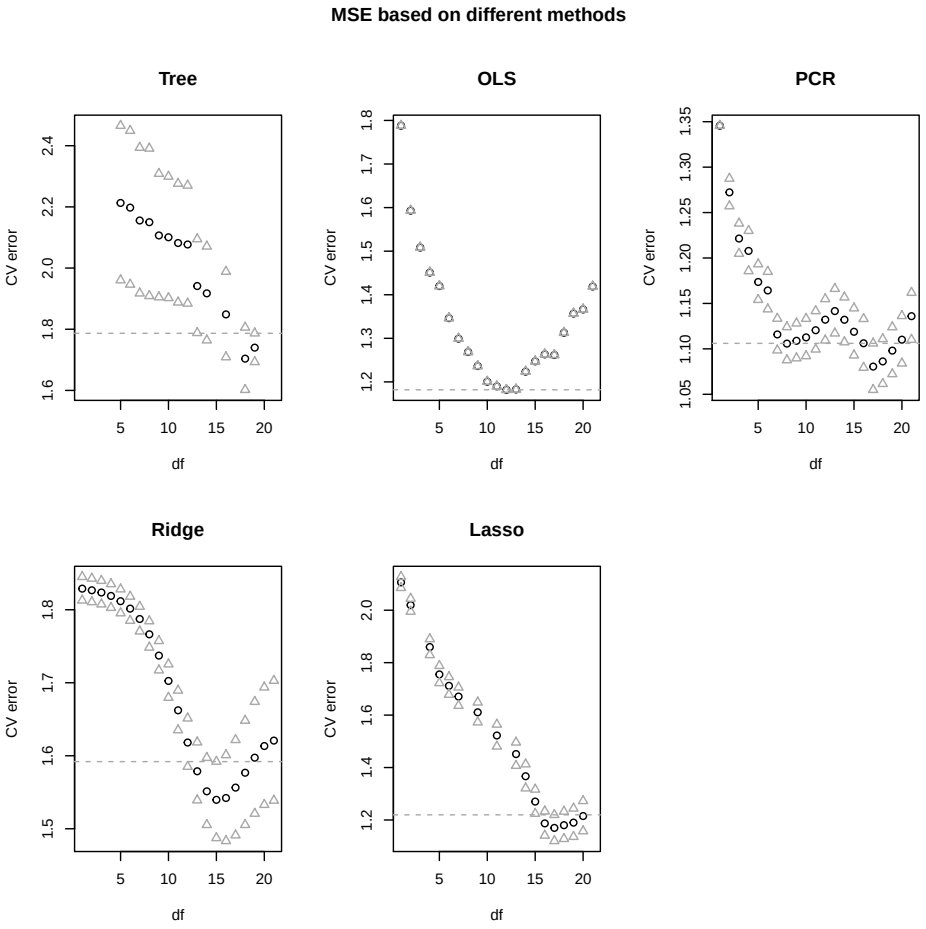
```r
cv_ridge = function(data){
  x = as.matrix(data[,-1])
  y = data[,1]
  cv.out = cv.glmnet(x,y,alpha=0,type.measure="mse")
  MSE = cv.out$cvm
  lambda = cv.out$lambda
  m = -log(lambda)
  mapped_m = (m-min(m))/(max(m)-min(m))*20
  df = data.frame(mse=MSE,mapped_m=as.integer(mapped_m))
  y=c()
  for (i in 0:20){
    ind = which(df$mapped_m==i)
    y=c(y,mean(df$mse[ind]))
  }
  return (list(y=y,x=unique(df$mapped_m) ))
}


## # Lasso
cv_lasso = function(data){
  x = as.matrix(data[,-1])
  y = data[,1]
  cv.out = cv.glmnet(x,y,alpha=1,type.measure="mse")
  MSE = cv.out$cvm
  deg = cv.out$nzero
  df = data.frame(mse=MSE,deg=deg)
  y=c()
  for (i in 0:20){
    ind = which(df$deg==i)
    y=c(y,mean(df$mse[ind]))
  }
  return (list(y=y,x=unique(deg)))
}
```

Now, I write a plot function in order to see the difference performances of the methods used. Basically I compute the different methods B times and then I compute the means and the standard errors of the distributions of the cross validated errors for each model. I parallelize everything in order to make it much faster. Hoever, I do not parallelize the tree method for a little difficulty I don't want to extend myself on.

```
library(parallel)
plot_cv = function(method, B=100, same_y = FALSE){
  data = makedata()
  if (method=="Tree"){
    CV=t(sapply(1:B, function(i,...)
    {a=cv_tree(data)$y;return(a)}))
  }
  else{
    ncores = detectCores()-1
    cl = makeCluster(ncores)
    clusterEvalQ(cl,c(library(pls),library(glmnet),library(tree)))
    clusterExport(cl,list("makedata","cv_pcr",
                          "cv_tree","cv_ridge","cv_ols",
                          "cv_lasso","cv_ols_one","glmnet"))

    if (method=="PCR"){R = parSapply(cl, 1:B, function(i,...)
    {a=cv_pcr(data)$y; return(a)})}
    if (method=="Lasso"){R = parSapply(cl, 1:B, function(i,...)
    {a=cv_lasso(data)$y; return(a)})}
    if (method=="Ridge"){R = parSapply(cl, 1:B, function(i,...)
    {a=cv_ridge(data)$y; return(a)})}
    if (method=="OLS"){R = parSapply(cl, 1:B, function(i,...)
    {a=cv_ols(data)$y; return(a)})}

    CV = matrix(unlist(R), nrow = B, byrow = T)
    stopCluster(cl)
  }

  avg = apply(CV,2,function(x){
    if (all(is.nan(x))){NA}
    else{mean(x[!is.nan(x)])}})
  se = apply(CV,2,function(x){
    if (all(is.nan(x))){NA}
```

```
    else{sd(x[!is.nan(x)])}})
  if (same_y){
    plot(avg,ylim=c(0.8,2),
         xlab="df",ylab="CV error",main=method)
  }
  else{
    plot(avg,ylim=c(min(avg-se,na.rm = T),max(avg+se,na.rm = T)),
         xlab="df",ylab="CV error",main=method)
  }
  points(avg+se,col="darkgrey",pch=2)
  points(avg-se,col="darkgrey",pch=2)
  abline(h = min(avg+se,na.rm = T), lty = 2, col="darkgrey")
}
```

```
# Different scales
par(mfrow=c(2,3),oma=c(0,0,3,0))
plot_cv("Tree")
plot_cv("OLS")
plot_cv("PCR")
plot_cv("Ridge")
plot_cv("Lasso")
title("MSE based on different methods",outer=T)
par(mfrow=c(1,1))
```



OLS won because of the data generating process. The data satisfies the OLS assumptions, i.e the orthogonality of the variables. We can see that the best model is for *df* around 10. However, for the trees, df=20 gives the best model. This is because trees have a very low bias (but a high variance). They perform poorly compared to the other methods here. The ridge regression and the lasso perform well here, although the L2 penalty here (ridge) gives better results than the L1 (lasso) because since the data is gaussian, it makes more sense to maximize the posterior with a gaussian prior than a double exponential prior.

2 Correlated explanatory variables

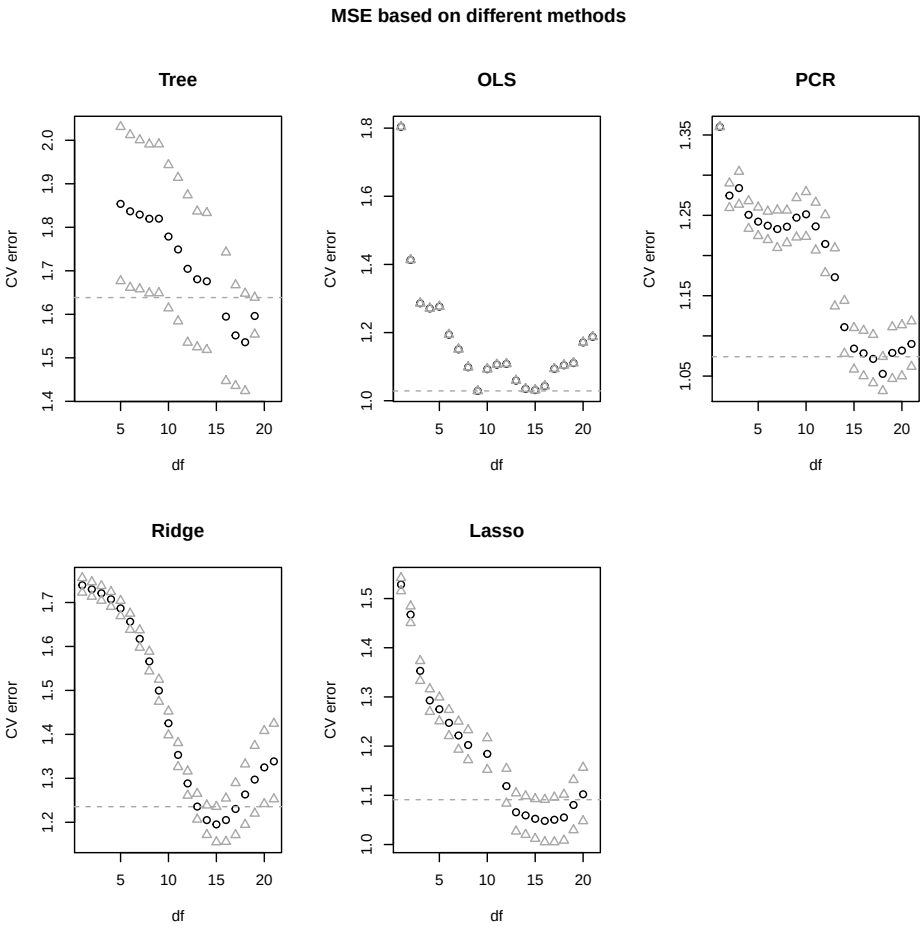
I change the *makedata* function in order to have correlated explanatory variables.

```
rm(makedata)

gen_corr = function(n = 100,
                    r = 0.05,
                    n_replications = 3){
  G = matrix(r,n,n) + diag(1-r,n,n)
  chol_G = chol(G)
  epsilons = matrix(rnorm(n*n_replications),n,n_replications)
  epsilons = as.data.frame(chol_G %*% epsilons)
  return(epsilons)
}

makedata = function(n=100,p=20){
  data1 = gen_corr(r=.1, n=100)
  data2 = gen_corr(r=.3, n=100)
  data3 = gen_corr(r=.5, n=100)
  data4 = gen_corr(r=.7, n=100)
  data5 = gen_corr(r=.9, n=100)
  data6 = matrix(rnorm(5*n),ncol=5, nrow=n)
  X = as.matrix(cbind(data1,data2,data3,data4,
                data5,data6))
  names(X) = 1:20
  exps=seq(-1,-2.5,length=p)
  beta=rep(0,p)
  beta=exp(exps)
  Y=.5+X%*%beta+rnorm(n)
  data = data.frame(Y,X)
  colnames(data)=c("Y",letters[1:20])
  switch=sample(20)+1
  data=data[,c(1,switch)]
  names(beta)=names(data)[switch]
  return(data)
}
```

If we make the explanatory variables correlated, the OLS assumptions

are no longer true.

```
# Different scales
par(mfrow=c(2,3),oma=c(0,0,3,0))
plot_cv("Tree")
plot_cv("OLS" )
plot_cv("PCR" )
plot_cv("Ridge")
plot_cv("Lasso")
title("MSE based on different methods",outer=T)
par(mfrow=c(1,1))
```



3 Discussion

A few commments:

- 1 - In the first simulation, the backward selection with OLS do have the best results since the data is generated with 20 independent random variables, and this satisfies the OLS assumption. In the second simulation, PCR behaves better than OLS since the variables are not orthogonal. Indeed, the orthogonalization of the features force the OLS assumptions to be true.

- 2 - Ridge regression shrinks all directions, but unlike lasso it does not explicitely set any variable to be zero. Lasso has a heavy penalty here compared to ridge and OLS. In OLS, the variables kept are the orthogonal ones, i.e. the 5 orthogonal generated at the end of the data set and the 5 others which have nothing to do with the other. In Ridge we keep almost all of them and in Lasso the best df is around 5.

- 3 - Principal components regression only takes some orthogonal components to do the regression, and discards the rest. In other words it only takes some components of the orthonormalized basis of the feature set. Lasso applies a L1 penalty and Ridge a L2 penalty. The L1 penalty tends to shrink the feature coefficients to exactly zero. On the other hand, Ridge will include all the p features in the model.

- 4 - For tree based method it is very different. Since the trees have a very low bias, the more the features the better. However, the splits strongly depend on the data set, which means they have high variance. This is why they behave poorly. In order to reduce the variance of the algorithm, we can take i.i.d versions of trees by bootstrapping the set of features before fitting a tree and then averaging over all the trees: this leads to random forests.