Overview and Setup

In this experiment we were to predict on a set of video transcoding time dataset. Data contained predictors ranging from duration to various features peratining to frame sizes, frame/bit rate and processor categories on the source and destination devices. After removing the ID column we examined the data by looking for singularities as well as linear dependence and redundancy issues. Variable mp below shows the unique levels in each column and from that it is apparent that b_size is singular, so we removed that. Additionally we know that i+p=frames and similarly i_size+p+size=frame_szie so we removed “frames” and “frame_size” right away.

In order to further investigate the data we looked at the cor matrix and used a cor>0.98 threshold to remove columns that showed high levels of collinearity. From the cor matrix width/height and o_height/o_width had cor>0.98 so we removed the width and o_width columns as well. From literature survey we know that the video arrival rate has a Poisson distribution which has log dependence.We experimented with examining the log relation of utime with other predictors.Additionally we used a log scale for the continuous predictors which were all in the form of bit/frame rates and left the other categorical predictors as is. Using log scale for utime as well as the continuous predictors didnt seem to affect our ols RMSE value but it slightly improved our Ridge and Lasso RMSE values.

Additionally to double-check our above assumptions, we did a subset selection on the data set and looked at the BIC values. The minimum index BIC value also confirmed the predictors we had experimentally selected. Using the corrected dataset we looked at a forward model and used that model to compare to Ridge and Lasso predictions. We noticed that the forward model gave warnings that it found dependencies in the matrix. We tried revisiting the dataset with plotting variable against each other as well as against utime but could not find other linear dependencies. As discussed perhaps using a VIF function would have been more effective in identifying additional linear dependences.

We used the matrix.model function to create dummies for non-numeric categorical data in our dataset (codec and o_codec) since the glmnet function cannnot handle non-numeric entries. We divide the dataset into two sections: One for training the model and the other for evaluations.

The following is our code along with the results:

```r
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-5
```

```
library(leaps)



utime.dat<-read.csv(file="transcoding_mesurment.tsv",sep="\t",header=TRUE)

## remove the id column and create a matrix.model for processing categorcial data with glmnet
  utime.dat=utime.dat[-1]
  nrow=dim(utime.dat)[1]
  ncol=dim(utime.dat)[2]
  fit.all=lm(utime~.,utime.dat)
  # summary(fit.all)

## create Dummy variables for codec and o_codec to be used in regsubsets function
  for(level in unique(utime.dat$codec)){
    utime.dat[paste("codec", level, sep = "")] <- ifelse(utime.dat$codec == level, 1, 0)
  }

  for(level in unique(utime.dat$o_codec)){
    utime.dat[paste("o_codec", level, sep = "")] <- ifelse(utime.dat$o_codec == level, 1, 0)
  }
  #head(utime.dat)


#Extract the numeric columns
  nums=sapply(utime.dat,is.numeric)
  utime.dat<-utime.dat[,nums]
  ncol=dim(utime.dat)[2]

## Find the column vector that is singular
  mp=matrix(0,1,ncol)
  for (i in 1:ncol){
    mp[1,i]=length(unique(utime.dat[,i]))
  }
   #mp

##b_size is singular so we remove that
  utime.dat=utime.dat[-12]
  ncol=dim(utime.dat)[2]

## Look at the correlation matrix
  # d<-cor(utime.dat)
  # collinear = matrix(0, ncol, ncol)
  # rownames(collinear) = colnames(utime.dat)
  # colnames(collinear) = colnames(utime.dat)
  # for (j in 1:(ncol-1)){
  #   for (i in (j+1):ncol){
  #     if (abs(d[i,j]) > 0.95){
  #       collinear[i,j] = 999
  #     } else if (abs(d[i,j])> 0.75){
  #       collinear[i,j] = 777
  #     }
  #   }
  # }
  #
  # diag(collinear)=NA
  # collinear

  ## from the collinearity matrix height/width, o_height/o_width, p/frames, p_size/size seem to
 have very high correlation
  ##Remove the redundant & singular columns
    drops<-c("frames","frame_size","width","o_width","b_size","o_codech264")
    df<-utime.dat[,!(names(utime.dat) %in% drops)]
    #head(df)

  ##Use log scale for continuous columns to follow a poisson distribution
    df$duration = log(df$duration)
    df$bitrate = log(df$bitrate)
    df$framerate = log(df$framerate)
    df$i = log(df$i)
    df$p = log(df$p)
    df$i_size = log(df$i_size)
    df$p_size = log(df$p_size)
    df$umem = log(df$umem)
    df$utime = log(df$utime)

  #Create the Model matrix for glmnet
    x.utime.gl=model.matrix(utime~.,df)[,-1]
    y.utime=utime.dat$utime
    ncol.new=length(x.utime.gl[1,])

##Divide the data into train and test samples

  set.seed(2000)
  index=sample (1: 2, nrow,replace=TRUE)
  train.index=which(index==2)
  test.index=which(index==1)

  ##Matrix model data for glmnet
  train.data=x.utime.gl[train.index,]
  test.data=x.utime.gl[test.index,]
  y_train=utime.dat$utime[train.index]
  y_test=utime.dat$utime[test.index]

  ## Training/test Data frame for lm
  df.train.data=df[train.index,]
  df.test.data=df[test.index,]


##Forward subset fit
  subsets.log.model = regsubsets(utime ~ .,     data = df.train.data, nvmax = 31)  ##y~. so we h
ave 2^11 combinations
```

```
## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax,
## force.in = force.in, : 1 linear dependencies found
## Reordering variables and trying again:
```

```
  forward.log.model     = regsubsets(utime ~ (.)^2, data = df.train.data, nvmax = 218, method = 
"forward")
```

```
## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax,
## force.in = force.in, : 31 linear dependencies found
```

```
## Reordering variables and trying again:
```

```
## Warning in rval$lopt[] <- rval$vorder[rval$lopt]: number of items to
## replace is not a multiple of replacement length
```

```
  summary.subsets.model=summary(subsets.log.model)
  summary.forward.model=summary(forward.log.model)
  bic.subsets.model=summary.subsets.model$bic
  bic.forward.model=summary.forward.model$bic
  index.subsets=which.min(bic.subsets.model)
  index.forward.min=which.min(bic.forward.model)

coef.forward=coef(forward.log.model,index.forward.min)
coef.subset=coef(subsets.log.model,index.subsets)

v1                = toString(names(coef(subsets.log.model, index.subsets))[-1])
v2                = gsub(pattern = ", ",  replacement = " + ", x = toString(v1))
subset.formula  = as.formula(paste("utime ~ ", v2, sep = ""))

u1                = toString(names(coef(forward.log.model, index.forward.min))[-1])
u2                = gsub(pattern = ", ",  replacement = " + ", x = toString(u1))
forward.formula  = as.formula(paste("utime ~ ", u2, sep = ""))

### create three models: ridge, lasso, and ols.
### Extracting the coef from the ridge and lasso requires specifying
###  the value of lambda at which the lowest cv happened.
###

ridge_model = cv.glmnet(x = train.data, y = log(y_train), alpha = 0)  #cv stands for cross valid
ate
ridge_model$lambda.min
```

```
## [1] 0.08246587
```

```
ridge_coef  = coef(ridge_model, s = ridge_model$lambda.min)  ## need to specify the min lambda

lasso_model = cv.glmnet(x = train.data, y = log(y_train), alpha = 1)
lasso_model$lambda.min
```

```
## [1] 0.001343988
```

```
lasso_coef  = coef(lasso_model, s = lasso_model$lambda.min)

ols_model   = lm(forward.formula, df.train.data)
ols_coef=coef(forward.log.model,index.forward.min)

subset_model   = lm(subset.formula, df.train.data)

###
### Predict on the test data next.
###

rmse = function(a,b) { sqrt(mean((a-b)^2)) }
#subset_rmse   = rmse(y_test , exp(predict(subset_model, data=df.test.data )))
ols_rmse   = rmse(y_test , exp(predict(ols_model, data=df.test.data )))
```

```
## Warning in a - b: longer object length is not a multiple of shorter object
## length
```

```
ridge_rmse = rmse(y_test , exp(predict(ridge_model, newx = test.data, s =
ridge_model$lambda.min)))
lasso_rmse = rmse(y_test , exp(predict(lasso_model, newx = test.data, s =
lasso_model$lambda.min)))

#subset_rsme
ols_rmse
```

```
## [1] 22.62847
```

```
ridge_rmse
```

```
## [1] 9.254621
```

```
lasso_rmse
```

```
## [1] 8.774756
```

Results and Discussions

As shown above our ols_rmse is 22.6 284 while Ridge_rmse is 9.2546 and Lasso_rmse is 8.7747. In our case Ridge and Lasso had better rmse. This is somewhat expected given that both Ridge and Lasso aim at eliminating dependencies by minimizing or bring the dependent coefficient close to zero. To further improve the results we could have looked at VIF to quantify the degree of multicollinearity before feeding the data in our forward model. Additionally we could have looked at other relationhips between the utime and other predictors but given the scope of this assignment we did not proceed to do so.