Linear Regression Analysis: Regression Case Study

Neerja Doshi, Sri Santhosh Hari, Ker-Yu Ong, Nicha Ruchirawat

```
knitr::opts_chunk$set(echo = TRUE, eval = F, cache = TRUE, tidy.opts=list(width.cutoff=65),tidy=TRUE)

# Clear working environment
rm(list=ls())

# Load requisite packages
if(!require("tidyverse")){install.packages("tidyverse", repos = "http://cran.us.r-project.org")}
if(!require("glmnet")){install.packages("glmnet", repos = "http://cran.us.r-project.org")}
if(!require("ggplot2")){install.packages("ggplot2", repos = "http://cran.us.r-project.org")}
if(!require("magrittr")){install.packages("magrittr", repos = "http://cran.us.r-project.org")}
if(!require("gridExtra")){install.packages("gridExtra", repos = "http://cran.us.r-project.org")}
if(!require("olsrr")){install.packages("olsrr", repos = "http://cran.us.r-project.org")}
if(!require("data.table")){install.packages("data.table", repos = "http://cran.us.r-project.org")}
if(!require("MASS")){install.packages("MASS", repos = "http://cran.us.r-project.org")}
if(!require("car")){install.packages("car", repos = "http://cran.us.r-project.org")}
if(!require("perturb")){install.packages("perturb", repos = "http://cran.us.r-project.org")}

# Turn off scientific notation
options(scipen=999)
```

Part I: Explanatory Modelling

Task 0: Exploratory Data Analysis and Data Cleaning

```
# Load Raw Data
rawDF <- read.csv("/Users/santhoshhari/Documents/Coursework/LinearRegression/IowaHousing/Data/housing.t
```

Handling NA Values

Below, we compute the number and percentage of `NA`s per variable in the dataset (only variables with atleast 1 `NA` will be reported)

```
NA_columns <- colnames(rawDF)[unique(which(is.na(rawDF), arr.ind = T)[,2])]

NA_count <- rawDF %>%
            dplyr::select(NA_columns) %>%
            summarise_all(funs(sum(is.na(.)))) %>%
            gather(key = "Variable", value = "num_na", everything()) %>%
            arrange(desc(num_na))

NA_count %<>% mutate(perc_na = paste(round(num_na/nrow(rawDF),4)*100,"%"))
colnames(NA_count) <- c("**Variable**", "**Number of NA**", "**Percentage of NA**")
row.names(NA_count) <- NULL
knitr::kable(NA_count, caption = "\\label{tab:NACount} Variable NA Count and Percentage",
format.args = list(big.mark = ','))
```

Refer to data dictionary and replace meaningful NAs with appropriate values.

```
# Create a copy of rawDF to be our working data frame
housingDF <- rawDF

# Update NAs with 0s for applicable fields
levels(housingDF$PoolQC) <- c("0",levels(housingDF$PoolQC))
housingDF$PoolQC[is.na(housingDF$PoolQC)] <- "0"
levels(housingDF$MiscFeature) <- c("0", levels(housingDF$MiscFeature))
housingDF$MiscFeature[is.na(housingDF$MiscFeature)] <- "0"
levels(housingDF$Alley) <- c("0", levels(housingDF$Alley))
housingDF$Alley[is.na(housingDF$Alley)] <- "0"
levels(housingDF$Fence) <- c("0", levels(housingDF$Fence))
housingDF$Fence[is.na(housingDF$Fence)] <- "0"
levels(housingDF$FireplaceQu) <- c("0", levels(housingDF$FireplaceQu))
housingDF$FireplaceQu[is.na(housingDF$FireplaceQu)] <- "0"
levels(housingDF$GarageType) <- c("0", levels(housingDF$GarageType))
housingDF$GarageType[is.na(housingDF$GarageType)] <- "0"
levels(housingDF$GarageFinish) <- c("0", levels(housingDF$GarageFinish))
housingDF$GarageFinish[is.na(housingDF$GarageFinish)] <- "0"
levels(housingDF$GarageQual) <- c("0", levels(housingDF$GarageQual))
housingDF$GarageQual[is.na(housingDF$GarageQual)] <- "0"
levels(housingDF$GarageCond) <- c("0", levels(housingDF$GarageCond))
housingDF$GarageCond[is.na(housingDF$GarageCond)] <- "0"
levels(housingDF$BsmtExposure) <- c("0", levels(housingDF$BsmtExposure))
housingDF$BsmtExposure[is.na(housingDF$BsmtExposure)] <- "0"
levels(housingDF$BsmtFinType2) <- c("0", levels(housingDF$BsmtFinType2))
housingDF$BsmtFinType2[is.na(housingDF$BsmtFinType2)] <- "0"
levels(housingDF$BsmtQual) <- c("0", levels(housingDF$BsmtQual))
housingDF$BsmtQual[is.na(housingDF$BsmtQual)] <- "0"
levels(housingDF$BsmtCond) <- c("0", levels(housingDF$BsmtCond))
housingDF$BsmtCond[is.na(housingDF$BsmtCond)] <- "0"
levels(housingDF$BsmtFinType1) <- c("0", levels(housingDF$BsmtFinType1))
housingDF$BsmtFinType1[is.na(housingDF$BsmtFinType1)] <- "0"
```

We then re-check the count and percentage of NAs per variable left in the dataset. This will give us the details of variables with actual missing data.

```
NA_columns <- colnames(housingDF)[unique(which(is.na(housingDF), arr.ind = T)[,2])]

NA_count <- housingDF %>%
            dplyr::select(NA_columns) %>%
            summarise_all(funs(sum(is.na(.)))) %>%
            gather(key = "Variable", value = "num_na", everything()) %>%
            arrange(desc(num_na))

NA_count %<>% mutate(perc_na = paste(round(num_na/nrow(housingDF),4)*100,"%"))
colnames(NA_count) <- c("**Variable**", "**Number of NA**", "**Percentage of NA**")
row.names(NA_count) <- NULL
knitr::kable(NA_count, caption = "\\label{tab:NACount1} Variable NA Count and Percentage(after replacin
format.args = list(big.mark = ','))
```

We impute NAs in these variables with

- mean of the data, for continuous variables (LotFrontage)
- median of the data, for discrete variables (GarageYrBlt)

- mode of the data, for categorical variables (MasVnrType, Electrical)

```
# Function to get mode of data
getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}

# Impute NAs
housingDF$LotFrontage[is.na(housingDF$LotFrontage)] <- mean(housingDF$LotFrontage, na.rm = T)
housingDF$GarageYrBlt[is.na(housingDF$GarageYrBlt)] <- median(housingDF$GarageYrBlt, na.rm=T)
housingDF$MasVnrType[is.na(housingDF$MasVnrType)] <- getmode(housingDF$MasVnrType)
housingDF$MasVnrArea[is.na(housingDF$MasVnrArea)] <- 0
housingDF$Electrical[is.na(housingDF$Electrical)] <- getmode(housingDF$Electrical)

# Convert MSSubClass to factor
housingDF$MSSubClass <- factor(housingDF$MSSubClass)
housingDF$MoSold <- factor(housingDF$MoSold)
```

Since Masonry veneer area (`MasVnrArea`) is directly related to `MasVnrType`, we impute for area based on the mode of `MasVnrType`.

Exploratory Data Visualization

Create functions for data visualization

```
plotHist <- function(data_in, i) {
  data <- data.frame(x=data_in[[i]])
  p <- ggplot(data=data, aes(x=factor(x))) +
       stat_count() +
       xlab(colnames(data_in)[i]) +
       theme_light() +
       theme(axis.text.x = element_text(angle = 90, hjust =1))
  return (p)
}

doPlots <- function(data_in, fun, ii, ncol=3) {
  pp <- list()
  for (i in ii) {
    p <- fun(data_in=data_in, i=i)
    pp <- c(pp, list(p))
  }
  do.call("grid.arrange", c(pp, ncol=ncol))
}

plotDen <- function(data_in, i){
  data <- data.frame(x=data_in[[i]], SalePrice = data_in$SalePrice)
  p <- ggplot(data= data) +
       geom_line(aes(x = x), stat = 'density', size = 1,alpha = 1.0) +
       xlab(paste0((colnames(data_in)[i]), '\n', 'Skewness: ',
                   round(skewness(data_in[[i]], na.rm = TRUE), 2))) +
       theme_light()
  return(p)
}
```

```
plotCorr <- function(data_in, i){
  data <- data.frame(x = data_in[[i]], SalePrice = data_in$SalePrice)
  p <- ggplot(data, aes(x = x, y = SalePrice)) +
       geom_point(na.rm = TRUE) +
       geom_smooth(method = lm ) +
       xlab(paste0(colnames(data_in)[i], '\n', 'R-Squared: ',
                round(cor(data_in[[i]], data$SalePrice, use = 'complete.obs'), 2))) +
       theme_light()
  return(suppressWarnings(p))
}
```

With our clean dataset, we perform exploratory data visualization of the distribution of key measures such as volume and sale price of houses by what we hypothesize to be key predictor variables.

To begin with, we check the distribution of sale prices in different neighborhoods using a box-plot.

```
housingDF %>%
  dplyr::select(Neighborhood, SalePrice) %>%
  ggplot(aes(factor(Neighborhood), SalePrice)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust =1)) +
  xlab('Neighborhoods')

housingDF %>%
  dplyr::select(LandSlope, Neighborhood) %>%
  arrange(Neighborhood) %>%
  group_by(Neighborhood, LandSlope) %>%
  summarize(Count = n()) %>%
  ggplot(aes(Neighborhood, Count)) +
  geom_bar(aes(fill = LandSlope), position = 'dodge', stat = 'identity')+
  theme(axis.text.x = element_text(angle = 90, hjust =1))
```

```
doPlots(housingDF, fun = plotHist, ii = c(3,6,7,10), ncol = 2)
doPlots(housingDF, fun = plotHist, ii = c(8,9,11,12), ncol = 2)
```

```
num_var <- names(housingDF)[which(sapply(housingDF, is.numeric))]
housing_numeric <- housingDF[num_var]
correlations <- cor(housing_numeric[,-1])
highcorr <- c(names(correlations[,'SalePrice'])[which(correlations[,'SalePrice'] > 0.5)], names(correla
data_corr <- housingDF[highcorr]
doPlots(data_corr, fun = plotCorr, ii = 1:10)
```

Task 1. Building the Explanatory Model

Testing for Influential Points

Having dealt with the `NA`s in our dataset, we use the `model.matrix()` function from the `glmnet` package to convert each categorical variable into an appropriate set of binary indicators: for a categorical variable that takes k levels, `model.matrix()` produces k-1 binary indicators. We then reappend our response vector `SalePrice` to the resulting wide design matrix `designDF` to create `workingDF`, which includes both the converted predictors and response variables.

```
designDF <- model.matrix(SalePrice ~ ., data = housingDF)[,-1]
designDF <- as.data.frame(designDF)
workingDF <- cbind(designDF, SalePrice = housingDF$SalePrice)
```

In looking for influential points, we leverage the `OLSRR` package to test observations for influence according to the DFFITS diagnostic. We do this by first fitting a saturated model on `workingDF` and then calling `ols_dffits_plot()` on it.

```
# Fit a saturated OLS model
ols_model <- lm(SalePrice ~ ., data = workingDF)
ols_dffits_plot(ols_model)
# Identify threshold t for points of influence
n = nrow(workingDF)
p = ncol(workingDF - 1) # remove response var
t = 2*sqrt(p/n)
# Calculate dffits and influential points
df <- dffits(ols_model)
influential_points <- which(abs(df)>t)
```

Note that according to the criterion of threshold t = 2*sqrt(n/p), the DFFITS plot shows a large number of influential observations. We then plot the standardized and studentized residuals to check these observations for being outliers and/or points of leverage respectively.

```
# Create df of studentized residuals
student_resid <- as.data.frame(rstudent(ols_model))
student_resid <- setDT(student_resid, keep.rownames = TRUE)[]
colnames(student_resid) <- c('ix', 'resid')
# Plot of residuals
plot(student_resid$ix, student_resid$resid, col = ifelse(workingDF$Id %in% influential_points, "red", "
abline(h = t, col = 'red')
abline(h = -t, col = 'red')
```

```
# Create df of standardized residuals
standard_resid <- as.data.frame(rstandard(ols_model))
standard_resid <- setDT(standard_resid, keep.rownames = TRUE)[]
colnames(standard_resid) <- c('ix', 'resid')
# Plot of Residuals
plot(standard_resid$ix, standard_resid$resid, col = ifelse(workingDF$Id  %in% influential_points, "red"
abline(h = t, col = 'red')
abline(h = -t, col = 'red')
```

Our plots of studentized and standardized resdiuals indicate that all observations are both points of leverage and outliers. We remove them from our dataset and recreate the saturated OLS model below:

```
# Remove influential points
workingDF <- filter(workingDF, !Id %in% influential_points)

# Recreate OLS model after removing influential points
ols_model <- lm(SalePrice ~ ., data = workingDF)
```

For the purposes of variable selection, we refer to the saturated OLS model created above and perform stepwise model selection according to BIC criterions.

```
model_bic <- step(ols_model, k = log(nrow(workingDF)), direction = 'backward', trace = F)

save(model_bic, file = "/Users/santhoshhari/Documents/Coursework/LinearRegression/IowaHousing/BIC_model
```

Per the BIC criterion, predictor variables signficant at the alpha = 0.01 level are considered.

```
# Load BIC
load("/Users/santhoshhari/Documents/Coursework/LinearRegression/IowaHousing/BIC_model.rda")
```

```
# find coefficients significant at the alpha = 0.01 level
bool_bic <- summary(model_bic)$coeff[-1,4] < 0.01
sig_var_bic <- names(bool_bic)[bool_bic == TRUE]
```

We now perform OLS regression with our subset of significant variables.

```
model_sig_formula <- as.formula(paste("SalePrice ~ ", paste(sig_var_bic, collapse = "+")))
model_sig <- lm(formula = model_sig_formula, data = workingDF)

# create a model with scaled Xs and Y for multicollinearity detection
model_sig_scaled <- lm(formula = model_sig_formula, data = as.data.frame(scale(workingDF)))
```

We check for multicollinearity in our model by checking for Variance Inflation Factors:

```
vif(model_sig_scaled)[(sqrt(vif(model_sig_scaled)) > 10) == TRUE]
```

We check for variables with VIF values $>$ threshold $= 10$. This tells us there is multicollinearity present in the dataset. We verify this by checking the Singular Value Criteria.

```
coll_out = colldiag(model_sig, scale = TRUE, center = FALSE, add.intercept = TRUE)
coll_out$condindx[(coll_out$condindx > 30)== TRUE]
coll_out = colldiag(model_sig, scale = TRUE, center = FALSE, add.intercept = TRUE)
```

We check if there are entries $>$ threshold $= 30$, and conclude if multicollinearity exists.

Based on results, we drop the `Garage Condition` variables since they are collinear with the Garage Quality variables.

```
# Drop GarageCondition variables since they are correlated with Garage Quality variables
drop = c('GarageCondEx','GarageCondFa', 'GarageCondGd', 'GarageCondPo' )
new_sig_var_bic = sig_var_bic [! sig_var_bic %in% drop]
```

We then rerun the model and recheck for multicollinearity using VIFS:

```
new_model_formula <- as.formula(paste("SalePrice ~ ", paste(new_sig_var_bic, collapse = "+")))
new_model <- lm(formula = new_model_formula, data = workingDF)

# create a model with scaled Xs and Y for multicollinearity detection
new_model_scaled <- lm(formula = new_model_formula, data = as.data.frame(scale(workingDF)))

# check for MC
vif(new_model_scaled)[(sqrt(vif(new_model_scaled)) > 10) == TRUE]
```

Using our rule of thumb, we conclude that there is no more multicollinearity in our model since there are no VIF values > 10. We then move on to validating the linearity and normality assumptions of our model by checking residuals.

```
# Residual plots
res = resid(new_model) # residuals
stdres = rstandard(new_model) # standardized residuals

# QQ Plot of Residuals - for normality
qqnorm(stdres, main = "QQ Plot of Standardized Residuals", xlab = "Normal Scores", ylab = "Standardized
qqline(stdres)

# Test for normality
ks.test(scale(res),rnorm(length(workingDF)))
```

```
# Plot of residuals vs. fitted values - for linearity
plot(workingDF$SalePrice, res, main = "Plot of Residuals vs. Sale Price", xlab = "Sale Price", ylab = "
abline(h=c(0,0), col = 'red')
abline(v=350000, col = 'blue')
```

Check QQ plot for linearity and residual plot for normality and apply boxcox transformation as needed.

```
boxcox(new_model)
```

We see that lambda = 1 is not captured in the 95% CI of lambdas. This means a transformation is necessary. We choose lambda = ~ 0.5 since this is an interpretable transformation value. We then apply a square root transformation to Sale Price, refit the model, and revalidate our model assumptions with residual plots as above.

```
# sqrt Transformation
sqrt_model_formula <- as.formula(paste("sqrt(SalePrice) ~ ", paste(new_sig_var_bic, collapse = "+")))
sqrt_model <- lm(formula = sqrt_model_formula, data = workingDF)

res_sqrt = resid(sqrt_model) # residuals
stdres_sqrt = rstandard(sqrt_model) # standardized residuals

ks.test(scale(res_sqrt),rnorm(length(workingDF)))

# Plot of Residuals from Log Model vs. Fitted Values
plot(sqrt(workingDF$SalePrice), res_sqrt, main = "Plot of Square Root Model Residuals vs. Squaer Root o
abline(h=c(0,0), col = 'red')
abline(v=350000, col = 'blue')

# Normal Probability Plot of Residuals from Log Model
qqnorm(stdres_sqrt, main = "QQ Plot of Standardized Square Root Model Residuals", xlab = "Normal Scores
qqline(stdres_sqrt)
```

We conclude that the variables included above are most relevant in determining a house's sale price.

Task 2: Making recommendations

```
#morty <- read.csv("/Users/booranium/usf/601_regression/project/Morty.txt", stringsAsFactors = T)
morty <- read.csv("/Users/santhoshhari/Documents/Coursework/LinearRegression/IowaHousing/Data/Morty.txt
morty <- morty[,-1]
# Replace Nulls
morty['PoolQC'] <- 'O'
morty['FireplaceQu'] <- 'O'
morty['Alley'] <- 'O'
# Transform morty data as per model object's requirement
new_data <- rbind(housingDF, morty)
new_data <- model.matrix(SalePrice ~ ., data = new_data)[,-1]
x_morty <- as.data.frame(tail(new_data,1))
new_sig_var_bic[which(new_sig_var_bic=="`RoofMatlTar&Grv`")] <- "RoofMatlTar&Grv"
new_sig_var_bic[which(new_sig_var_bic=="`Exterior1stWd Sdng`")] <- "Exterior1stWd Sdng"
new_sig_var_bic[which(new_sig_var_bic=="`Exterior2ndWd Sdng`")] <- "Exterior2ndWd Sdng"
sig_var_morty <- x_morty[,new_sig_var_bic]
```

Plot significant coefficients, to help in recommendations.

```
coef <- data.frame(coef.name = names(coef(sqrt_model)),
           coef.value = matrix(coef(sqrt_model)))

# exclude the (Intercept) term
coef <- coef[-1,]
coef <- arrange(coef,-coef.value)
imp_coef <- head(coef,25)
ggplot(imp_coef) +
    geom_bar(aes(x=reorder(coef.name,coef.value),y=coef.value), stat="identity") +
    coord_flip() +
    ggtitle("Coefficents in the Model") +
    theme(axis.title=element_blank())
```

Test improvement ideas using the coeficient plot

```
morty <- read.csv("/Users/santhoshhari/Documents/Coursework/LinearRegression/IowaHousing/Data/Morty.txt
# Replace Nulls
morty['PoolQC'] <- '0'
morty['FireplaceQu'] <- '0'
morty['Alley'] <- '0'
# 1st recommendation
morty['Exterior1st'] <- 'BrkFace'
# 2nd recommendation
morty['BsmtExposure'] <- 'Av'
# 3rd recommendation
morty['MasVnrType'] <- 'Stone'

morty <- morty[,c(-1,-2)]
new_data <- as.data.frame(rbind(housingDF[,-1], morty))
new_data <- model.matrix(SalePrice ~ ., data = new_data)
x_morty <- as.data.frame(tail(new_data,1))
new_sig_var_bic[which(new_sig_var_bic=="`RoofMatlTar&Grv`")] <- "RoofMatlTar&Grv"
new_sig_var_bic[which(new_sig_var_bic=="`Exterior1stWd Sdng`")] <- "Exterior1stWd Sdng"
new_sig_var_bic[which(new_sig_var_bic=="`Exterior2ndWd Sdng`")] <- "Exterior2ndWd Sdng"
sig_var_morty <- x_morty[,new_sig_var_bic]

round(predict.lm(sqrt_model,sig_var_morty,interval = 'confidence', level=0.95)^2,0)
```

Part II: Predictive Modelling

For comparing the prediction accuracy, we use 4 models - 1. Ridge Regression 2. Lasso Regression 3. Elastic Net

Transformations done on the data before building models - 1. Imputing NAs 2. Creating dummy variables 3. Removing influential points

Train and test sets: 75% of the data forms the train set 25% of the data forms the test set

```
x <- workingDF[,-1]
x$SalePrice <- log(workingDF[,ncol(workingDF)])
#y_log = log(y)
#train/test
set.seed(121)
```

```
train <- sample(1:nrow(x), 3 * nrow(x)/4)
test <- (-train)
```

Ridge:

```
set.seed(121)
x <- workingDF[,2:ncol(workingDF)-1]
y <- workingDF$SalePrice

#train/test
y.train <- y[train]
y.test <- y[test]

ridge<-cv.glmnet(as.matrix(x[train, ]), y.train, alpha=0)
plot(ridge)
best.lambda <- ridge$lambda.min
best.lambda
abline(v = log(best.lambda), col = 'blue', lwd = 2)

ridge.model.train <- glmnet(as.matrix(x[train, ]), y.train, alpha = 0, lambda = best.lambda)


ridge.pred <- predict(ridge.model.train, s = best.lambda, newx = as.matrix(x[test,]))
mspe.ridge <- mean((ridge.pred - y.test)^2)

coef_ridge <- coef(ridge.model.train)
length(coef_ridge[coef_ridge != 0])
```

Lasso:

```
set.seed(121)

x <- workingDF[,2:ncol(workingDF)-1]
y <- workingDF$SalePrice

lasso <-cv.glmnet(as.matrix(x[train, ]), y.train, alpha=1)
plot(lasso)
best.lambda <- lasso$lambda.min
best.lambda
abline(v = log(best.lambda), col = 'blue', lwd = 2)

lasso.model.train <- glmnet(as.matrix(x[train, ]), y.train, alpha = 1, lambda = best.lambda)


lasso.pred <- predict(lasso.model.train, s = best.lambda, newx = as.matrix(x[test,]))
mspe.lasso <- mean((lasso.pred - y.test)^2)
mspe.lasso
#coef(lasso)
coef_lasso <- coef(lasso.model.train)
length(coef_lasso[coef_lasso!= 0])
```

Elastic Net:

```
set.seed(121)
x <- workingDF[,2:ncol(workingDF)-1]
y <- workingDF$SalePrice
```

```r
#find best alpha
alphalist<-seq(0,1,by=0.1)
elasticnet<-lapply(alphalist, function(a){cv.glmnet(as.matrix(x[train, ]), y.train, alpha=a)})
mse <- list()
for (i in 1:11) {
  mse <- c(mse,min(elasticnet[[i]]$cvm) )
  print(min(elasticnet[[i]]$cvm))
  print(i)}
which.min(unlist(mse))
alpha_min = alphalist[which.min(unlist(mse))]

#find best lambda for best alpha
cv.out <- cv.glmnet(as.matrix(x[train, ]), y.train, alpha = alpha_min)

plot(cv.out)
best.lambda <- cv.out$lambda.min
best.lambda
abline(v = log(best.lambda), col = 'blue', lwd = 2)

#train EN
EN.model.train <- glmnet(as.matrix(x[train, ]), y.train, alpha = alpha_min, lambda = best.lambda)

EN.pred <- predict(EN.model.train, newx = as.matrix(x[test,]))
mspe.EN <- mean((EN.pred - y.test)^2)
coef_elastic <- coef(EN.model.train)
length(coef_elastic[coef_elastic!= 0])
```

Consolidate MSPE values

```r
MSPE <- data.frame(Ridge = mspe.ridge, Lasso = mspe.lasso, EN = mspe.EN)
```