Predicting House Prices

Alonso Quijano

The goal of this project is to use regularization techniques to predict house prices. Ridge, lasso and, ultimately, elastic net models were used to train the algorithm.

Data

The data used in this project was downloaded from Kaggle, a website that promotes data science education. It contains data of the price and house characteristics of residential homes in Ames, Iowa, including neighborhood, lot size in square feet, building type, construction type, number of rooms and so.

We can have a look at the distribution of prices, the average price in the sample is around 180k dollars.

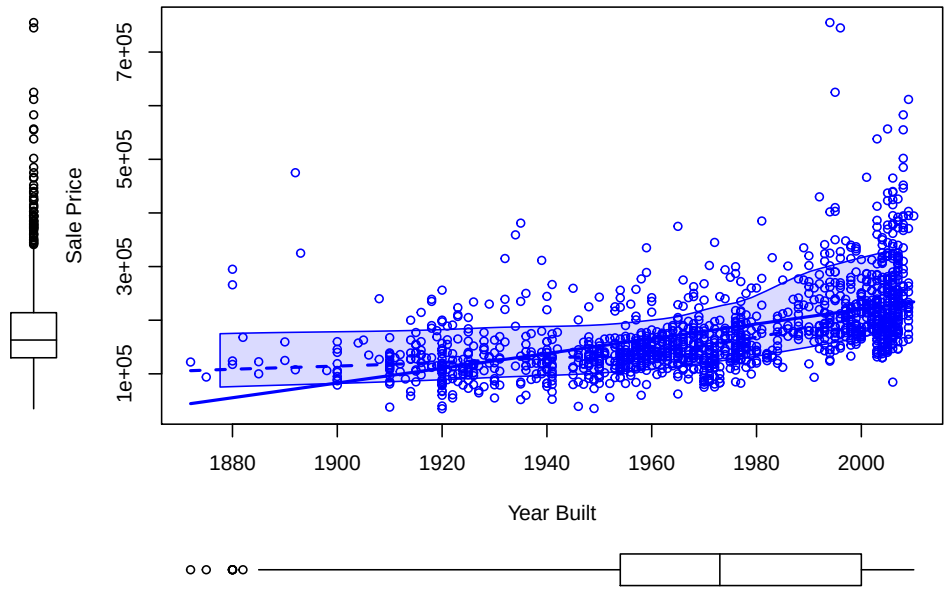
```
hist(houses$SalePrice, freq = FALSE, main = "Histogram of house prices")
```


```
summary(houses$SalePrice)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   34900  129975  163000  180921  214000  755000
```

We can also look at the relationship between price and the year a house was build.

```
# relationship between price and year a house was build
scatterplot(SalePrice ~ YearBuilt, data=houses,  xlab="Year Built", ylab="Sale Price", grid=FALSE)
```



Methods

Ridge and lasso regressions

Ridge regression and lasso regression will be used to train the algorithm. Ridge and lasso regressions are regularization techniques that can help deal with multicollinearity. When collinearity is high, least squares estimates are unbiased but have high variance. Thus, they tend to have a large out of sample error. Ridge and lasso regressions deal with this problem by introducing some bias to the model but largely decreasing its variance. In practice, these models work by shrinking the parameters; therefore, preventing multicollineary and reducing model complexity (overfitting).

Ridge and lasso estimators are obtained by minimizing the sum of squared errors plus a penalty. Ridge regression, also known as L2 regularization, is done by minimizing:

$$\hat{\beta}^R = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right) + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

In contrast lasso regression, also known as L1 regularization, minimizes the following loss function:

$$\hat{\beta}^L = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right) + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

How much the parameters are shrunk depends on λ. A larger λ will result is a larger shrinkage. The key difference between ridge and lasso regressions is that lasso regression can shrink coefficients all the way to zero, while ridge regression can only shrink coefficients asymptotically to zero. Thus, lasso regression does not only reduce variance but can actually be used for variable selection.

Elastic net

Lasso regression's advantage of variable selection is also its main disadvantage. When variables are highly correlated, lasso regression would tend to choose only one variable and ignore the rest. Elastic net combines

both ridge and lasso regressions. Elastic net is a penalized model that includes both L1 and L2 penalties for training. How much weight is given to each penalty is determined by α. The elastic net penalty is as follows:

$$\Omega(\beta) = \alpha\|\beta\|_1 + (1-\alpha)\|\beta\|_2^2$$

If $\alpha = 0.5$, equal weight is assigned to both ridge and lasso penalties. When $\alpha = 1$, all weight is given to the lasso penalty. In contrast, when when $\alpha = 0$ all weight is given to the ridge penalty.

The variables

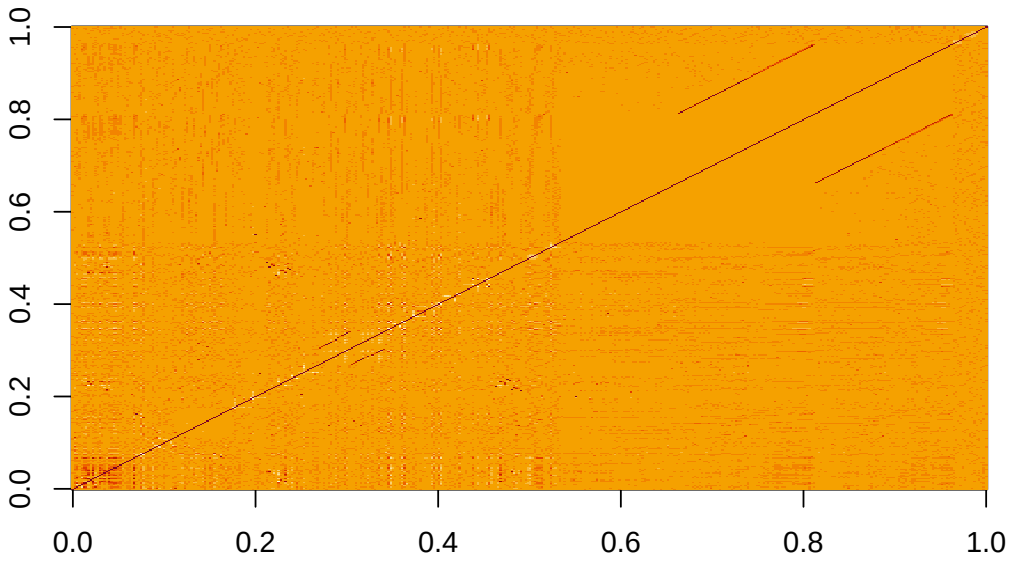
The dataset contains 1460 observations and 80 variables, including the house prices. First, we remove all the variables with empty observations. Then, we dichotomize categorical variables by making the data wide and filling the values with 1s and 0s.

```
houses <- houses[ , colSums(is.na(houses)) == 0] # removes columns with NAs

# dichotomize function
dichotomize <- function(data,var) {
  for(x in var) {
    l <- length(data)
    data$v <- 1
    data <- spread(data, key = x, value = v, fill = 0, sep = "_")
    data <- data[, -l] # eliminates one column for colinearity
  }
  return(data)
}

# variables to dichotomize
binvar <- names(houses)[sapply(houses, FUN = is.character)] # character variable names
binvar <- c(binvar, c("MSSubClass", "OverallQual", "OverallCond","YearBuilt",
                      "YearRemodAdd", "YrSold", "MoSold"))
houses <- dichotomize(houses, binvar)

# heatmap to observe correlations
image(cor(houses))
```



After dichotomizing the categorical variables the total number of explanatory variables increased to 398. This dataset now becomes highly dimensional. To avoid overfitting, we will need to use regularization techniques to shrink the parameters.

Splitting the data

2/3rds of the data will be used for training and 1/3 of the data will be used for testing. Before that, house prices are transformed to log prices. Taking logs means that errors in predicting expensive houses and cheap houses will affect the result equally.

```
dim(houses) # includes Id
```

```
## [1] 1460  400
```

```
x <- houses[, -which(colnames(houses) %in% c("Id", "SalePrice"))]; x <- as.matrix(x)
y <- houses["SalePrice"]
y <- log(y) # log of price

# 2/3rds of the data will be used for training and 1/3 of the data will be used for testing
set.seed(25)
train_rows <- sample(1:nrow(houses), 0.66*nrow(houses))
x.train <- x[train_rows, ]
x.test <- x[-train_rows, ]

y.train <- y[train_rows]
y.test <- y[-train_rows]
dim(x.train)
```

```
## [1] 963 398
```

Model training

We will use the `cv.glmnet` function from the `glmnet` package. The "cv" in `cv.glmnet` means we are going to use cross validation to find optimal values for λ. By default the `cv.glmnet` function uses 10-fold cross validation.

Ridge regression

By setting `alpha=0`, we conduct a ridge regression. `type.measure="mse"` means we will use mean square error as the loss function, and `family="gaussian"` tells `cv.glmnet` we will use linear regression. In summary, we will use a linear regression model with ridge penalty using 10-fold cross validation to find optimal values for λ.

```
# ridge regression
set.seed(25)
alpha0.fit <- cv.glmnet(x = x.train, y = y.train, type.measure="mse", alpha=0, family="gaussian")
```

To test the algorithm, we will use the fitted model `alpha0.fit` and `lambda.1se` as the optimal λ, which resulted in the simplest model and was within 1 standard error of the λ that has the smallest sum.

```
# predicting y with ridge
alpha0.predicted <- predict(alpha0.fit, s=alpha0.fit$lambda.1se, newx=x.test)
```

The resulting mse of the ridge regression model is:

```
mean((alpha0.predicted - y.test)^2)
```

```
## [1] 0.03307831
```

Lasso

We set `alpha=1`. Now, we will use a linear regression model with lasso penalty using 10-fold cross validation to find optimal values for λ.

```
# lasso regression
set.seed(25)
alpha1.fit <- cv.glmnet(x = x.train, y = y.train, type.measure="mse", alpha=1, family="gaussian")
alpha1.predicted <- predict(alpha1.fit, s=alpha1.fit$lambda.1se, newx=x.test)
```

The resulting mse of the lasso regression model is:

```
mean((alpha1.predicted - y.test)^2)
```

```
## [1] 0.03247707
```

Elastic net

We set `alpha=0.5` and assing half the weight to the ridge penaly and half the weight to the lasso penalty.

```
# elastic net (50% lasso 50% ridge)
alpha0.5.fit <- cv.glmnet(x.train, y.train, type.measure="mse", alpha=0.5, family="gaussian")
alpha0.5.predicted <- predict(alpha0.5.fit, s=alpha0.5.fit$lambda.1se, newx=x.test)
```

The resulting mse of the lasso regression model is:

```r
mean((y.test - alpha0.5.predicted)^2)
```

```
## [1] 0.0333773
```

To find the optimal value of α, we will use 10-fold cross validation. We create a loop to test different values of alpha.

```r
# model training with different alphas
list.of.fits <- list() # creates a list to store the parameters of each model
set.seed(25)

for (i in 0:10) {
  fit.name <- paste0("alpha", i/10)
  list.of.fits[[fit.name]] <-
    cv.glmnet(x.train, y.train, type.measure="mse", alpha=i/10,
              family="gaussian")
}
```

```r
# model testing
results <- data.frame()
for (i in 0:10) {
  fit.name <- paste0("alpha", i/10)

  # y hat
  predicted <-
    predict(list.of.fits[[fit.name]],
            s=list.of.fits[[fit.name]]$lambda.1se, newx=x.test)

  mse <- mean((y.test - predicted)^2) # mean square error

  ## store the results
  temp <- data.frame(alpha=i/10, mse=mse, fit.name=fit.name)
  results <- rbind(results, temp)
}
```

The model that results in the lowest mean square error is an elastic net with $\alpha = 0.9$.

```r
results
```

```
##    alpha        mse fit.name
## 1    0.0 0.03307831   alpha0
## 2    0.1 0.03266980 alpha0.1
## 3    0.2 0.03392102 alpha0.2
## 4    0.3 0.03301137 alpha0.3
## 5    0.4 0.03380652 alpha0.4
## 6    0.5 0.03209880 alpha0.5
## 7    0.6 0.03185621 alpha0.6
## 8    0.7 0.03054727 alpha0.7
## 9    0.8 0.03409015 alpha0.8
## 10   0.9 0.03034467 alpha0.9
## 11   1.0 0.03133942   alpha1
```

```
plot(results$alpha, results$mse, xlab = "alpha", ylab = "mse", main = "Elastic net tuning")
```