ISE 5103 Intelligent Data Analytics

Homework 5 - Modeling

Daniel Carpenter & Sonaxy Mohanty

October 2022

Contents

Packages

```
# Data Wrangling
library(tidyverse)

# Modeling
library(MASS)
library(caret) # Modeling variants like SVM
library(earth) # Modeling with Mars

# Aesthetics
library(knitr)
library(cowplot)  # multiple ggplots on one plot with plot_grid()
library(scales)
library(kableExtra)
```

General Data Prep

Read Data

```
hd <- read.csv('housingData.csv') %>%

  # creates new variables age, ageSinceRemodel, and ageofGarage, and
  dplyr::mutate(age = YrSold - YearBuilt,
                ageSinceRemodel = YrSold - YearRemodAdd,
                ageofGarage = ifelse(is.na(GarageYrBlt), age, YrSold - GarageYrBlt)) %>%

  # removes the columns used in above the calculations
  dplyr::select(!c(Id,MSSubClass, MiscVal, YrSold ,
                   MoSold, YearBuilt, YearRemodAdd))


# Convert all character data to factor
hd[sapply(hd, is.character)] <-
  lapply(hd[sapply(hd, is.character)], as.factor)
```

Impute Missing Values with `PMM`

Make data set of `numeric` variables

```
hd.numericRaw <- hd %>%

  #selecting all the numeric data
  dplyr::select_if(is.numeric) %>%

  #converting the data frame to tibble
  as_tibble()
```

Make data set of `factor` variables

```
hd.factorRaw <- hd %>%

  #selecting all the numeric data
  dplyr::select_if(is.factor) %>%

  #converting the data frame to tibble
  as_tibble()
```

For each column with missing data, impute missing values with `PMM`

- Done with function `imputeWithPMM()` function
- Applies function via `dplyr` logic
- Note `seeImputation()` function to visualize the imputation from prior homework 4, not shown for simplicity in viewing

Create function to impute via `PMM`

```
imputeWithPMM <- function(colWithMissingData) {

  # Using the mice package
  suppressMessages(library(mice))

  # Discover the missing rows
  isMissing <- is.na(colWithMissingData)

  # Create data frame to pass to PMM imputation function from mic package
  df <- data.frame(x       = rexp(length(colWithMissingData)), # meaningless x to help show variation
                   y       = colWithMissingData,
                   missing = isMissing)

  # imputation by PMM
  df[isMissing, "y"] <- mice.impute.pmm( df$y,
                                         !df$missing,
                                          df$x)

  return(df$y)
}
```

Apply `PMM` function to numeric data containing null values

```
# Data to store imputed values with PMM method
hd.Imputed <- hd

# Which columns has Na's?
colNamesWithNulls <- colnames(hd.numericRaw[ , colSums(is.na(hd.numericRaw)) != 0])
colNamesWithNulls
```

```
## [1] "LotFrontage" "MasVnrArea"  "GarageYrBlt"
```

```
numberOfColsWithNulls = length(colNamesWithNulls)


# For each of the numeric columns with null values
for (colWithNullsNum in 1:numberOfColsWithNulls) {

  # The name of the column with null values
  nameOfThisColumn <- colNamesWithNulls[colWithNullsNum]

  # Get the actual data of the column with nulls
  colWithNulls <- hd[, nameOfThisColumn]

  # Impute the missing values with PMM
  imputedValues <- imputeWithPMM(colWithNulls)

  # Now store the data in the original new frame
  hd.Imputed[, nameOfThisColumn] <- imputedValues

  # Save a visualization of the imputation
  pmmVisual <- seeImputation(data.frame(y = colWithNulls),
                             data.frame(y = imputedValues),
                             nameOfThisColumn )

  fileToSave = paste0('OutputPMM/Imputation_With_PMM_', nameOfThisColumn, '.pdf')
  print(paste0('For imputation results of ', nameOfThisColumn, ', see ', fileToSave))
  ggsave(pmmVisual, filename = fileToSave,
         height = 11, width = 8.5)
}
```

```
## [1] "For imputation results of LotFrontage, see OutputPMM/Imputation_With_PMM_LotFrontage.pdf"
```

```
## [1] "For imputation results of MasVnrArea, see OutputPMM/Imputation_With_PMM_MasVnrArea.pdf"
```

```
## [1] "For imputation results of GarageYrBlt, see OutputPMM/Imputation_With_PMM_GarageYrBlt.pdf"
```

Factor Level Collapse - Create `Other` Bin for Columns over 4 Unique Values

```
hd.Cleaned <- hd.Imputed # For final cleaned data

# Get list of factors and the number of unique values
factorCols <- as.data.frame(t(hd.factorRaw %>% summarise_all(n_distinct)))

# We are going to factor collapse factor columns with more than 4 columns
# So there will be 4 of the original, and 1 containing 'other'
# This is the threshold
factorThreshold = 4

# Get a list of the factors we are going to collapse
colsWithManyFactors <- rownames(factorCols %>% filter(V1 > factorThreshold))

# Show a summary of how many factors will be collapsed
numberOfColsWithManyFactors = length(colsWithManyFactors)
paste('Before cleaning, there are', numberOfColsWithManyFactors, 'factor columns with more than',
      factorThreshold, 'unique values')
```

```
## [1] "Before cleaning, there are 14 factor columns with more than 4 unique values"
```

```
# Collapse the affected factors in the original data (the one that already has imputation)

## for each factor column that we are about to collapse
for (collapsedColNum in 1:numberOfColsWithManyFactors) {

  # The name of the column with null values
  nameOfThisColumn <- colsWithManyFactors[collapsedColNum]

  # Get the actual data of the column with nulls
  colWithManyFactors <- hd[, nameOfThisColumn]

  # lumps all levels except for the n most frequent
  hd.Cleaned[, nameOfThisColumn] <- fct_lump_n(colWithManyFactors,
                                               n=factorThreshold)
}

# Check to see if the factor lumping worked
factorColsCleaned <- t(hd.Cleaned %>%
                       select_if(is.factor) %>%
                       summarise_all(n_distinct))
paste('After cleaning, there are', sum(factorColsCleaned > factorThreshold, na.rm = TRUE),
      "columns with more than", factorThreshold, "unique values (omitting NA's)")
```

```
## [1] "After cleaning, there are 14 columns with more than 4 unique values (omitting NA's)"
```

Remove Outliers from Numeric Data

- Since there are so many outliers, we are only going to remove some outliers
- If you count the number of outliers by column, the 75% of columns contain less than 50 outliers.
- However, some contain up to 200. Since remove ALL outliers would reduce the size of the data to less than 300 observations, we are removing up to 50 per column.

```
hd.CleanedNoOutliers <- hd.Cleaned

# Remove up to 75% of the outliers in the data set
# this is the 3rd quartile of number of outliers.
k_outliers = 50
numOutliers = c() # to store the number of outliers per column

theColNames <- colnames(hd.Cleaned)

for (colNum in 1:ncol(hd.Cleaned)) {

  theCol <- hd.Cleaned[, colNum]
  nrowBefore = length(theCol)
  colName <- theColNames[colNum]


  # Only consider numeric
  if (is.numeric(theCol)) {

    # Identify the outliers in the column
    # Source: https://www.geeksforgeeks.org/remove-outliers-from-data-set-in-r/
    columnOutliers <- boxplot.stats(hd.CleanedNoOutliers[, colNum])$out
    numOutliers <- c(numOutliers, length(columnOutliers))

    # Now remove k outliers from the column
    if (length(columnOutliers) < k_outliers) {

      hd.CleanedNoOutliers  <- hd.CleanedNoOutliers %>%

        # If this syntax looks weird, it is just referencing a column in the
        # data set using dplyr piping. See below for more info:
        # https://stackoverflow.com/questions/48062213/dplyr-using-column-names-as-function-arguments
        # https://stackoverflow.com/questions/72673381/column-names-as-variables-in-dplyr-select-v-filt
        filter( !( get({{colName}}) %in% columnOutliers ) )
    }
  }
}
paste0('Of the columns with outliers, removed up to 75th percentile of num. outliers.')
```

```
## [1] "Of the columns with outliers, removed up to 75th percentile of num. outliers."
```

```
paste0('See that the 75th percentile of columns with outliers contain ',
       paste0(summary(numOutliers)[5]), ' outliers')
```

```
## [1] "See that the 75th percentile of columns with outliers contain 32.25 outliers"
```

1 (a) - OLS Model

1 (b) - PLS Model

1 (c) - LASSO Model

1 (d) - Model Variants

1 (d, i) - PCR Model

Model Setup

- Uses `numeric` data for Principal Component Analysis
- Then appends the `factor` data to the data *without* `NULL` *values*
- Finally, uses `stepAIC()` to best model data
- See interpretation at end

Get cleaned `numeric` and `factor data frames`

```
# After cleaning, two data sets that contain..

## Numeric data ----------------------------------------------------
hd.numericClean <- hd.Cleaned %>% select_if(is.numeric)

## Factors ---------------------------------------------------------
hd.factorClean  <- hd.Cleaned %>% dplyr::select(where(is.factor))

# Removing any columns with NA
removeColsWithNA <- function(df) {
  return( df[ , colSums(is.na(df)) == 0] )
}
hd.factorClean <- removeColsWithNA(hd.factorClean)

paste('Num. factor cols. removed due to null values:',
      ncol(hd.Cleaned %>% dplyr::select(where(is.factor)) ) - ncol(hd.factorClean) )
```

```
## [1] "Num. factor cols. removed due to null values: 16"
```

```
paste(ncol(hd.factorClean), 'factor cols. remain')
```

```
## [1] "22 factor cols. remain"
```

Perform PCA

```
# Principal component analysis on numeric data
pc.house <- prcomp(hd.numericClean %>% dplyr::select(-SalePrice), # do not include response var
                   center = TRUE, # Mean centered
                   scale  = TRUE  # Z-Score standardized
                   )

# See first 10 cumulative proportions
pc.house.summary <- summary(pc.house)
pc.house.summary$importance[, 1:10]
```

```
##                             PC1      PC2      PC3      PC4      PC5      PC6
## Standard deviation     2.656172 1.849617 1.598806 1.395032 1.170712 1.104997
## Proportion of Variance 0.227590 0.110360 0.082460 0.062780 0.044210 0.039390
## Cumulative Proportion  0.227590 0.337950 0.420400 0.483180 0.527390 0.566780
##                             PC7      PC8      PC9     PC10
## Standard deviation     1.062051 1.033024 1.006171 1.001662
## Proportion of Variance 0.036390 0.034420 0.032660 0.032370
## Cumulative Proportion  0.603170 0.637590 0.670250 0.702610
```

Now we choose number of PC's that explain 75% of the variation

- Note this threshold is just a judgement call. No significance behind 75%

```
cumPropThreshold = 0.75 # The threshold

numPCs <- sum(pc.house.summary$importance['Cumulative Proportion', ] < cumPropThreshold)
paste0('There are ', numPCs, ' principal components that explain up to ', cumPropThreshold*100,
       '% of the variation in the data')
```

```
## [1] "There are 11 principal components that explain up to 75% of the variation in the data"
```

```
chosenPCs <- as.data.frame(pc.house$x[, 1:numPCs])
```

Join on the factor data

```
df.pcr <- cbind(SalePrice = hd.numericClean$SalePrice, chosenPCs, hd.factorClean)
```

Fit the Model

- Linear model containing:
  - Principal components explaining 75% of variation in `numeric` data
  - Non-null `factor` data
  - *Predicted variable:* `log(SalePrice)`
- Then use `stepAIC()` to identify which variables are actually important for model

```
# Fit data using PC's, non-null factors
fit.pcr <- lm(log(SalePrice) ~ ., data = df.pcr)

# Reduce to only important variables
fit.pcrReduced <- stepAIC(fit.pcr, direction="both")
```

Model	Notes	Hyperparameters	RMSE	Rsquared
PCR	lm	N/A	0.1022416	0.9162997

```
# View results
summary(fit.pcrReduced)
```

```
##
## Call:
## lm(formula = log(SalePrice) ~ PC1 + PC2 + PC3 + PC4 + PC5 + PC7 +
##     PC8 + PC10 + MSZoning + LandContour + LotConfig + Condition1 +
##     BldgType + HouseStyle + RoofStyle + Exterior1st + ExterQual +
##     ExterCond + Foundation + CentralAir + KitchenQual + Functional +
##     PavedDrive, data = df.pcr)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.67514 -0.05941  0.00239  0.06691  0.31373
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)         11.800573   0.054621 216.045  < 2e-16 ***
## PC1                  0.097658   0.002399  40.713  < 2e-16 ***
## PC2                  0.005295   0.003427   1.545 0.122630
## PC3                 -0.053780   0.003392 -15.857  < 2e-16 ***
## PC4                 -0.020321   0.003653  -5.562 3.46e-08 ***
## PC5                  0.054058   0.003972  13.609  < 2e-16 ***
## PC7                  0.008111   0.003409   2.379 0.017541 *
## PC8                 -0.014628   0.003630  -4.030 6.02e-05 ***
## PC10                 0.007275   0.003701   1.966 0.049601 *
## MSZoningRH          -0.058488   0.040206  -1.455 0.146075
## MSZoningRL          -0.036574   0.020509  -1.783 0.074852 .
## MSZoningRM          -0.113959   0.022135  -5.148 3.19e-07 ***
## LandContourHLS       0.086694   0.027033   3.207 0.001386 **
## LandContourLow       0.002350   0.028827   0.082 0.935046
## LandContourLvl      -0.003108   0.018289  -0.170 0.865088
## LotConfigCulDSac     0.044479   0.015253   2.916 0.003627 **
## LotConfigInside      0.003833   0.009176   0.418 0.676293
## LotConfigother      -0.006192   0.019404  -0.319 0.749733
## Condition1Feedr      0.053849   0.024956   2.158 0.031195 *
## Condition1Norm       0.094366   0.020646   4.571 5.50e-06 ***
## Condition1RR         0.054456   0.029564   1.842 0.065793 .
## Condition1Other      0.027255   0.031592   0.863 0.388506
## BldgType2fmCon       0.034918   0.027746   1.258 0.208526
## BldgTypeDuplex       0.045787   0.026515   1.727 0.084524 .
## BldgTypeTwnhs       -0.050106   0.022153  -2.262 0.023934 *
## BldgTypeTwnhsE      -0.007736   0.015336  -0.504 0.614085
## HouseStyle1Story    -0.064499   0.015649  -4.122 4.09e-05 ***
## HouseStyle2Story    -0.011508   0.015288  -0.753 0.451810
## HouseStyleSLvl      -0.031953   0.020615  -1.550 0.121485
## HouseStyleOther     -0.055152   0.020277  -2.720 0.006648 **
## RoofStyleHip         0.016048   0.009437   1.701 0.089365 .
## RoofStyleother       0.104359   0.024928   4.186 3.10e-05 ***
## Exterior1stMetalSd   0.023801   0.012814   1.857 0.063552 .
## Exterior1stVinylSd   0.023160   0.011515   2.011 0.044581 *
## Exterior1stWd Sdng  -0.007375   0.013436  -0.549 0.583182
## Exterior1stOther     0.034323   0.011733   2.925 0.003522 **
## ExterQualAvg        -0.040288   0.011785  -3.419 0.000656 ***
## ExterQualBelowAvg   -0.124284   0.046998  -2.644 0.008317 **
## ExterCondAvg         0.018761   0.011746   1.597 0.110552
## ExterCondBelowAvg   -0.016695   0.032878  -0.508 0.611714
```

```
## FoundationCBlock     0.002493   0.014416   0.173 0.862759
## Foundationother      0.027201   0.025820   1.054 0.292378
## FoundationPConc      0.053152   0.016915   3.142 0.001729 **
## CentralAirY          0.055603   0.017321   3.210 0.001371 **
## KitchenQualAvg      -0.025460   0.010449  -2.437 0.015007 *
## KitchenQualBelowAvg -0.044179   0.025607  -1.725 0.084811 .
## FunctionalMaj2      -0.227058   0.062363  -3.641 0.000286 ***
## FunctionalMin1       0.022884   0.039389   0.581 0.561386
## FunctionalMin2       0.017675   0.038479   0.459 0.646087
## FunctionalMod       -0.004907   0.044754  -0.110 0.912723
## FunctionalTyp        0.087159   0.032315   2.697 0.007118 **
## PavedDriveP         -0.010503   0.025509  -0.412 0.680615
## PavedDriveY          0.047491   0.016346   2.905 0.003754 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1051 on 947 degrees of freedom
## Multiple R-squared:  0.9207, Adjusted R-squared:  0.9163
## F-statistic: 211.3 on 52 and 947 DF,  p-value: < 2.2e-16
```

View and Interpret Results

Please note all interpretations below are approximate, given the `stepAIC()` uses stochastic modeling.

Model performance evaluation:

- See that around 28 of the variables cannot be explained by random chance, with a probability of 90% or more (see significance codes above)
- Standard errors range from ± 1-5%, with average around 2%. Larger values may indicate higher uncertainty of the estimated coefficients.
- This model explains around 92% of the variation in the `log(SalePrice)`. See Adjusted R-Squared for reference.
- Note this model may exhibit selection bias, since the data excludes factor data with null values in the variable.
- This model would likely doe well for prediction of `log(SalePrice)`, given the small range of standard errors, high adjusted R squared, and number of significant variables. This model would obviously not do well for inference, given we are using principal components that mask the numeric data.

Practical significance evaluation:

- The principal components contribute positively about 20% of the sale price of the home
- Residential Medium Density (`MSZoningRM`) reduces the home price by around 12%, with a standard error of around 2%.
- If the exterior quality is below average (`ExterQualBelowAvg`), it reduces the home price by around 12%, with a standard error of around 5%.
- If the functionality of the home has 2 major deductions (`FunctionalMaj2`), it reduces the home price by around 20%, with a standard error of around 6%. While having typical functionality (`FunctionalTyp`) increases the home sale price by nearly 10%, with a standard error of 3%.
- See other coefficients of the data for other variables.

View Predicted vs. Actuals

Function to compare predicted vs. observed values

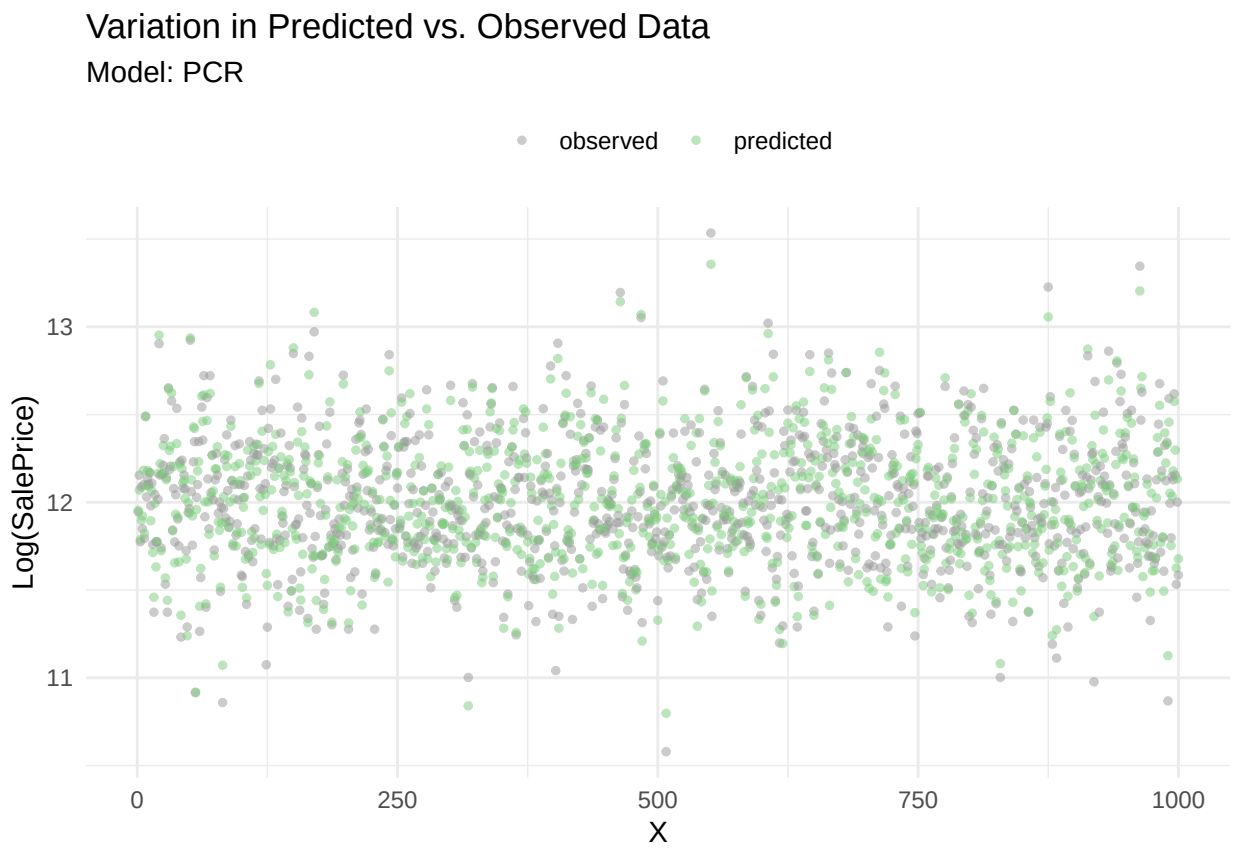
```
# Function to compare predicted vs. actual (observed) regression outputs
predictedVsObserved <- function(predicted, observed, modelName, outcomeName = 'Log(SalePrice)') {

  ## Create data set for predicted vs. actuals
  comparison <- data.frame(observed  = observed,
                           predicted = predicted) %>%

    # Row index
    mutate(ID = row_number()) %>%

    # Put in single column
    pivot_longer(cols      = c('observed', 'predicted'),
                 names_to  = 'metric',
                 values_to = 'value')


  # Plot --- Observed vs. Actuals across all variables in data
  variationScatter <- comparison %>%
    ggplot(aes(x     = ID,
               y     = value,
               color = metric
               )
           ) +
    geom_point(alpha = 0.5, size = 1) +

    labs(title = 'Variation in Predicted vs. Observed Data',
         subtitle = paste('Model:', modelName),
         x = 'X', y = outcomeName) +
    theme_minimal() + theme(legend.title = element_blank(),
                            legend.position = 'top') +
    scale_color_manual(values = c('grey60', 'palegreen3'))


  print(variationScatter)

  # Limit for x and y axis for scatter of predicted vs. observed
  axisLim = c( min(c(predicted, observed)), max(c(predicted, observed)) )


  # Simple comparison of data
  plot(x = observed,
       y = predicted,
       main = paste(modelName, 'Model - Actual (Observed) vs. Predicted\n'),
       xlab = paste('Observed Values -', outcomeName),
       ylab = paste('Predicted Values -', outcomeName),
       pch  = 16,
       cex  = 0.75,
       col  = alpha('steelblue3', 1/4),
       xlim = axisLim,
       ylim = axisLim
```

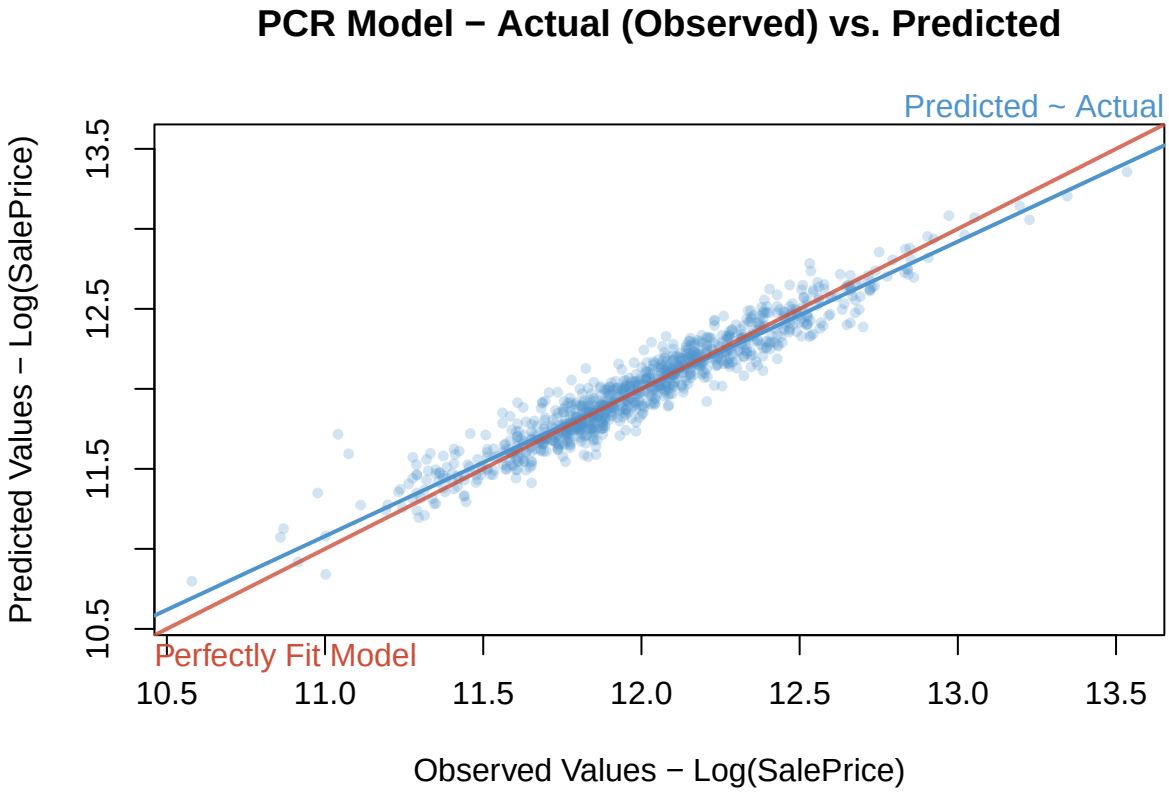
```
  )

  # Add the Predicted vs. actual line
  abline(lm(predicted ~ observed), col = 'steelblue3', lwd = 2)
  mtext('Predicted ~ Actual', side = 3, adj = 1, col = 'steelblue3')

  # Add line for perfectly fit model
  abline(0,1, col = alpha('tomato3', 0.8), lwd = 2)
  mtext('Perfectly Fit Model', side = 1, adj = 0, col = 'tomato3')
}
```

View results of the PCR Model

- See that the variation in the data is very closely resembled actual by changes in independent variables
- Implication? This model fits its own data well, but what is not know if it can predict out of sample data.
- Note that it the data (blue) deviates slightly from perfect line model (red), indicating that the model is slightly skewed from predicted and actual data.

```
# How do the predicted vs. Actuals Compare?
predictedVsObserved(observed  = log(df.pcr$SalePrice),
                    predicted = predict(fit.pcrReduced),
                    modelName = 'PCR')
```




PCR Model − Actual (Observed) vs. Predicted
Predicted ~ Actual
Perfectly Fit Model
Predicted Values − Log(SalePrice)
Observed Values − Log(SalePrice)
10.5
11.5
12.5
13.5
10.5
11.0
11.5
12.0
12.5
13.0
13.5

1 (d, ii) - SVR Model

Model Setup

```
ctrl <- trainControl(method  = "repeatedcv",
                     number  = 5, # 5 fold cross validation
                     repeats = 2  # 5 repeats
                     )

# The data (PMM imputed values, and only whole columns without NA. Does not omit outliers)
df.svm <- cbind(SalePrice = hd.numericClean$SalePrice,
                hd.numericClean, hd.factorClean)
```

Fit the Model

```
# Train and tune the SVM
fit.svm <- train(data = df.svm,
                 log(SalePrice) ~ .,
                 method     = "svmRadial",          # Radial kernel
                 tuneLength = 9,                    # 9 values of the cost function
                 preProc    = c("center","scale"), # Center and scale data
                 trControl  = ctrl)
```

View and Interpret Results

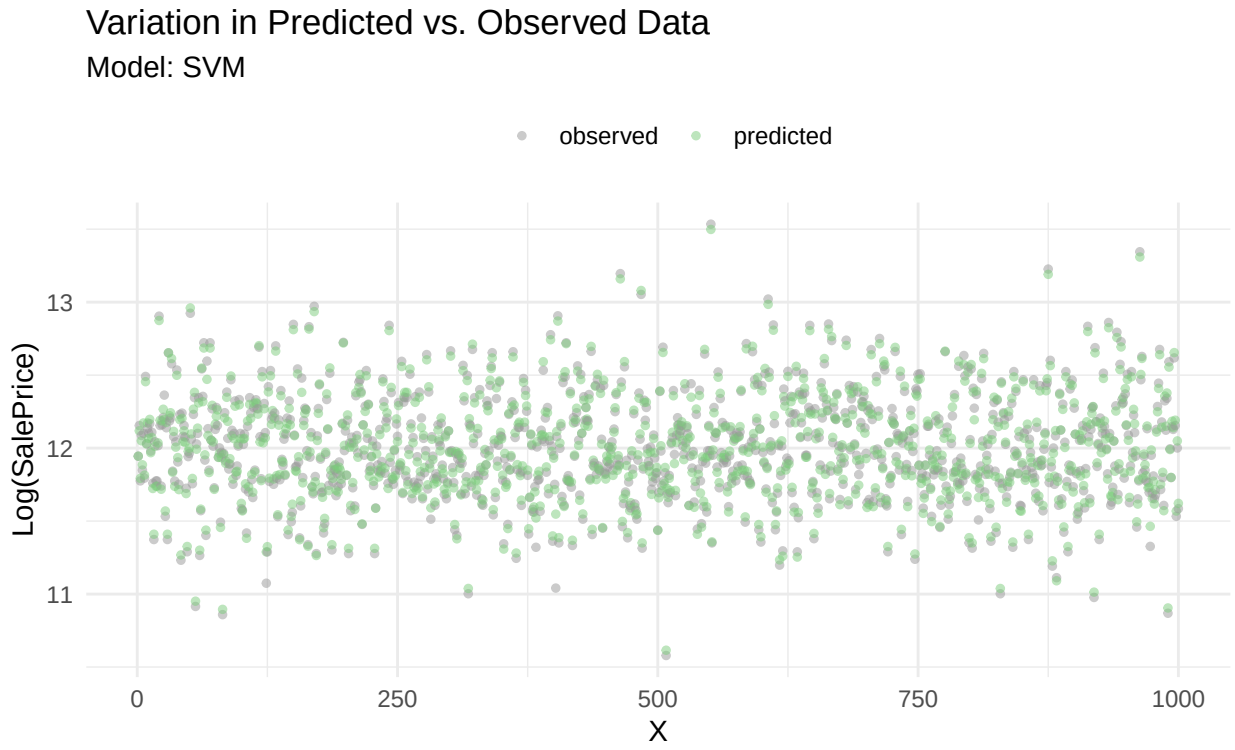
- Note all numbers mentioned below are approximate
- See that the R Squared of the model is around 0.86, and RMSE is 0.14
- See that the model predicts the data well.
- Also, note that the model predicts the data with less error than the linear model. See this from the RMSE or scatter plot of predicted values.

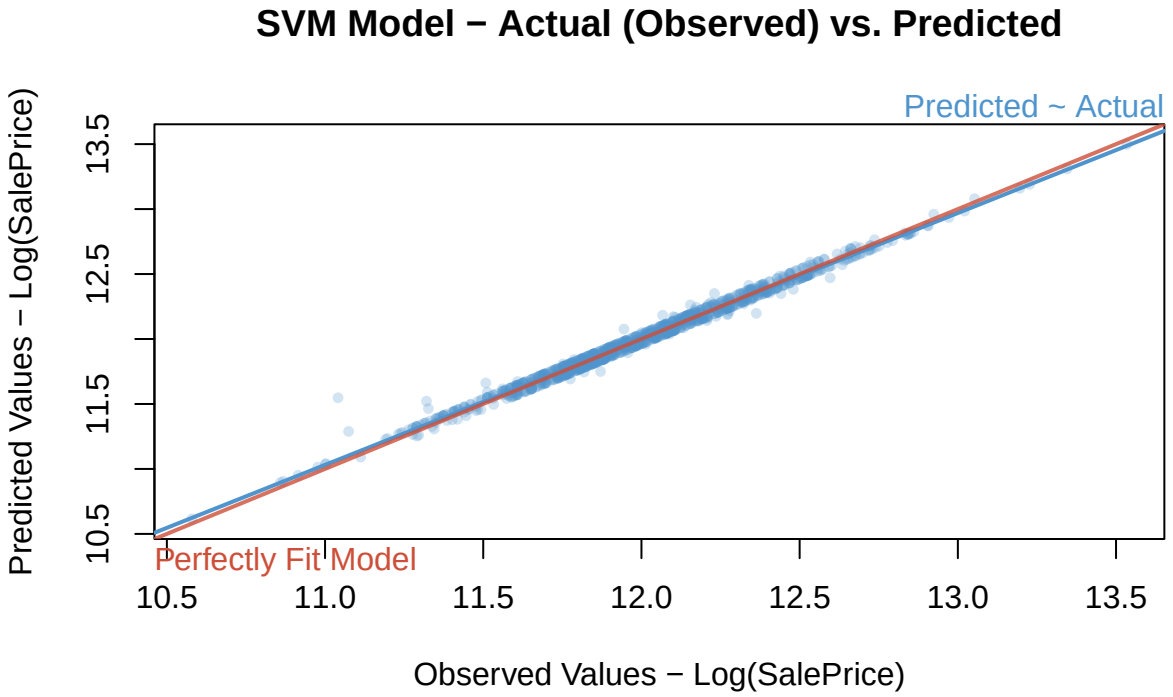
Model	Notes	Hyperparameters	RMSE	Rsquared
SVM	caret and svmRadial	C = 4 , Epsilon = 0.1	0.1369795	0.859835

```
# Final model?
fit.svm$finalModel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: eps-svr  (regression)
##  parameter : epsilon = 0.1  cost C = 4
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.00758001371246159
##
## Number of Support Vectors : 664
##
## Objective Function Value : -161.1988
## Training error : 0.012167
```

```
# How do the predicted vs. Actuals Compare?
predictedVsObserved(observed  = log(df.svm$SalePrice),
                    predicted = predict(fit.svm, df.svm),
                    modelName = 'SVM')
```





1 (d, iii) - MARS Model

Fit the Model

```
# Train and tune the MARS model
fit.mars <- train(data = df.svm, # note this is fine since data is the same for this model
                  log(SalePrice) ~ .,
                  method     = "earth",             # Radial kernel
                  tuneLength = 9,                   # 9 values of the cost function
                  preProc    = c("center","scale"), # Center and scale data
                  trControl  = ctrl
                 )
```

```
## Warning in preProcess.default(thresh = 0.95, k = 5, freqCut = 19, uniqueCut =
## 10, : These variables have zero variances: FunctionalMaj2
```

Model	Notes	Hyperparameters	RMSE	Rsquared
MARS	caret and earth	Degree = 1 , nprune = 17	0.1071991	0.9124208

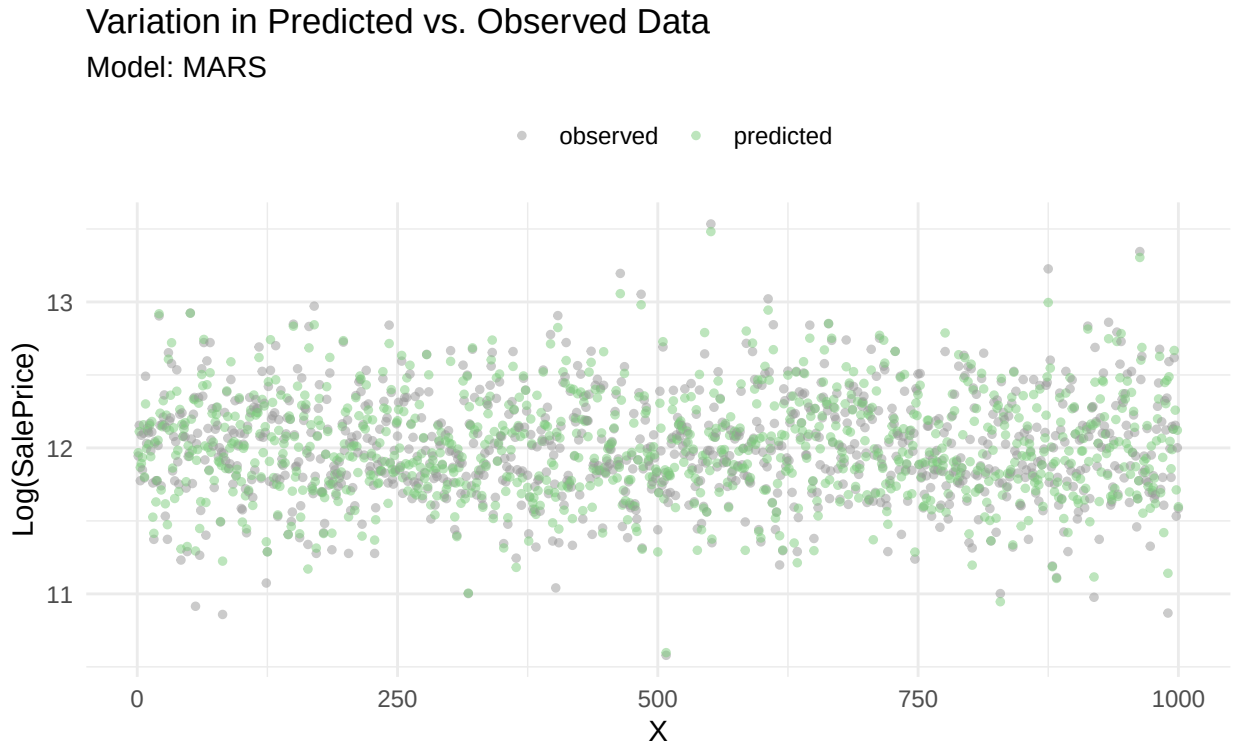
View and Interpret Results

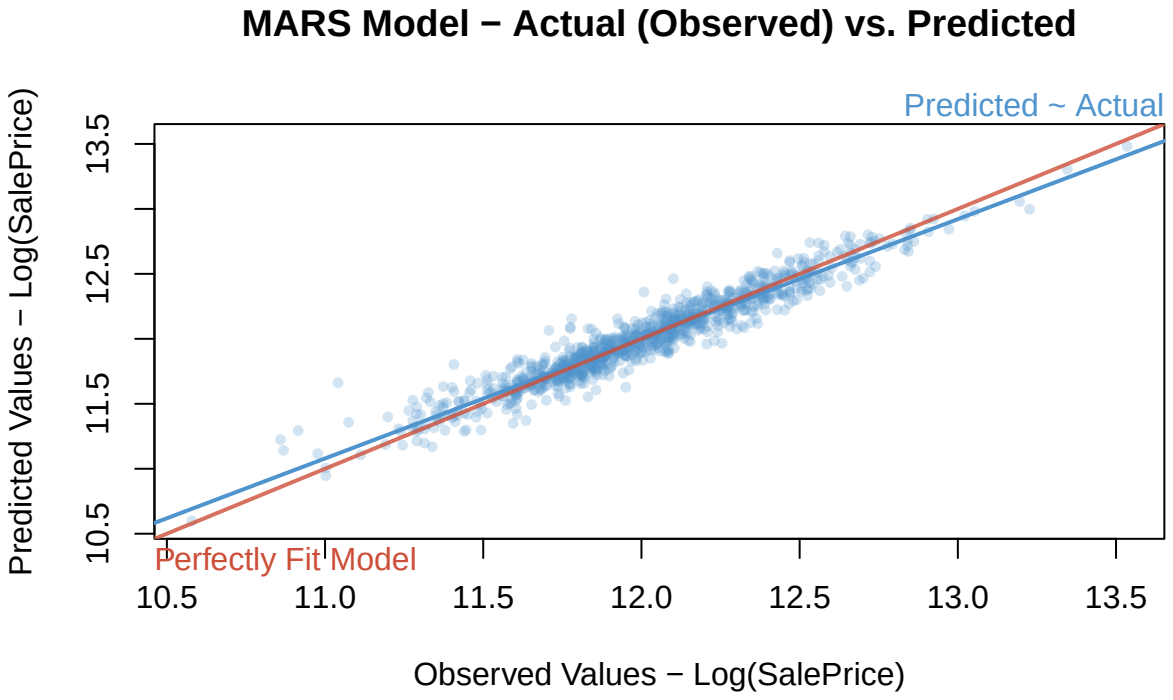
- See that the model overall performs very well, and in fact performs similarly to the PCR model (in terms of RMSE and Adjusted R Squared).
- Again, unsure if the model would do well for prediction of out of sample data, but fits this data extremely well.

```
# Final model?
fit.mars$finalModel
```

```
## Selected 17 of 21 terms, and 10 of 92 predictors (nprune=17)
## Termination condition: RSq changed by less than 0.001 at 21 terms
## Importance: GrLivArea, age, OverallQual, TotalBsmtSF, OverallCond, LotArea, ...
## Number of terms at each degree of interaction: 1 16 (additive model)
## GCV 0.011145    RSS 10.42157    GRSq 0.9155756    RSq 0.9208976
```

```
# How do the predicted vs. Actuals Compare?
predicted.mars = fit.mars[["finalModel"]][["fitted.values"]]
colnames(predicted.mars) <- 'predicted'

predictedVsObserved(observed  = log(df.svm$SalePrice),
                    predicted = predicted.mars,
                    modelName = 'MARS')
```


Variation in Predicted vs. Observed Data
Model: MARS
observed
predicted
Log(SalePrice)
13
12
11
0
250
500
750
1000
X


MARS Model − Actual (Observed) vs. Predicted
Predicted ~ Actual
Perfectly Fit Model
Predicted Values − Log(SalePrice)
10.5
11.5
12.5
13.5
10.5
11.0
11.5
12.0
12.5
13.0
13.5
Observed Values − Log(SalePrice)

Summary Table of Model Performance

Model	Notes	Hyperparameters	RMSE	Rsquared
PCR	lm	N/A	0.1022	0.9163
SVM	caret and svmRadial	C = 4 , Epsilon = 0.1	0.1370	0.8598
MARS	caret and earth	Degree = 1 , nprune = 17	0.1072	0.9124