Predicting House Prices with Various Machine Learning Models

Team ID A5

Justin Chiong and Haydne Golden

Introduction

Our objective for this project is to predict the house sale prices using a dataset that will feature various machine learning models such as linear regression, logistic regression using Gaussian and Poisson models, and XGBoost to see which model performs the best, and what the average sale prices would be based on their conditions, features, and appliances. The original dataset that we have chosen is about house sale prices and the attributes associated with each house. Some attributes listed are Lot Frontage, Lot Area, Year Built, and Building Type. Our data can be found on Kaggle and part of it was from a previous STAT 380 homework assignment that we plan to further enhance on.

House Sale Prices:

https://www.kaggle.com/competitions/psu-stat-380-001-house-prices/data

https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data?select=train.csv

Data and Methods

Our datasets contain 11460 rows of data combined giving us a large sample size to predict accurately with. Our data contains some “NA” values that we plan on either removing or reassigning through other methods. We have multiple variables regarding the functionality and listings of each house such as # of Bedrooms, Bathrooms, Lot Area, and many more. We are going to use R and RStudio with the libraries data.table, caret, Metrics, and ggplot2 to predict the House Sale Prices using the data and models listed above. As we mentioned before, we will be using various models such as linear regression, logistic regression (Gaussian and Poisson), and XGBoost to compare which models produce a better prediction and how they compare against each other. We will then calculate the evaluation metrics to figure out R-Squared value, Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE) for each of the ML models.

The first step we took in this project was to import the necessary libraries and import the raw data for our models. After this step we combine the two datasets to give our model more data to form better predictions. Once the data has been integrated, we split the data into training and testing data. We decided to go with a split of 70% training data and 30% testing data as we found this to be most effective when training our

model and preventing overfitting.

```
library(data.table)
library(caret)
library(Metrics)
rm(list = ls())

set.seed(123)
data1 <- fread('./project/volume/data/raw/Stat_380_train.csv')
kaggle_data <- fread('./project/volume/data/raw/kaggle_data.csv')

#Data Integration
#Create a datatable using only 17 columns from kaggle_data
data2 <- data.table(Id = kaggle_data$Id, LotFrontage = kaggle_data$LotFrontage,
data <- rbind(data1, data2)
data$SalePrice <- as.numeric(data$SalePrice)
fwrite(data, "./project/volume/data/processed/data.csv")
data <- fread('./project/volume/data/processed/data.csv')

#split data into training (70%) and testing (30%) sets
split1<- sample(c(rep(0, 0.7 * nrow(data1)), rep(1, 0.3 * nrow(data1))))
train <- data1[split1 == 0,]
test <- data1[split1== 1,]
```

Next we save our sales prices for both the testing and training sets so we can later reference them for our evaluation metrics. Then we create dummy variables using the training and testing data to be used in our prediction

```
#save actual test SalePrice
train_y <- train$SalePrice
test_y <- test$SalePrice
test$SalePrice <- 0

#dummy variables will split the variable into multiple variables containing boolean values
dummies <- dummyVars(SalePrice ~ ., data = train)

train <- predict(dummies, newdata = train)
train <- data.table(train)

test <- predict(dummies, newdata = test)
test <- data.table(test)

train$SalePrice <- train_y
```

Now that the dummy variables have been defined, we can use them to create our linear and logistic regression models. Once the models have been created, we save

them along with the dummy variables so that they may be used at a later date.

```
#creating the linear model
fit <- lm(SalePrice ~ ., data = train)
summary(fit)

#creating the gaussian logistic model
glm_fit <- glm(SalePrice ~ ., family = gaussian, data = train)
summary(glm_fit)

#creating the poisson logistic model
plm_fit <- glm(SalePrice ~ ., family = poisson, data = train)
summary(plm_fit)

saveRDS(dummies,"./project/volume/models/SalePriceRegression.dummies")
saveRDS(fit,"./project/volume/models/fit_lm.model")
saveRDS(glm_fit,"./project/volume/models/glm_fit.model")
saveRDS(plm_fit,"./project/volume/models/plm_fit.model")
```

The completed models are now used to predict the sales prices. We then create CSV files with these values.

```
#predicting the sale price using the linear model
test$Pred_SalePrice <- predict(fit, newdata = test)
submit_lm <- test[,.(Pred_SalePrice)]
submit_lm[is.na(submit_lm$Pred_SalePrice)] <- mean(train$SalePrice)
fwrite(test[,.(Id, SalePrice = submit_lm$Pred_SalePrice)], "./project/volume/data/processed/submit_lm.csv")

#predicting the sale price using the gaussian logistic model
test$Pred_result <- predict(glm_fit, newdata = test, type = "response")
submit_glm <- test[,.(Pred_result)]
submit_glm[is.na(submit_glm$Pred_result)] <- mean(train$SalePrice)
fwrite(test[,.(Id, result = submit_glm$Pred_result)], "./project/volume/data/processed/submit_glm.csv")

#predicting the sale price using the poisson logistic model
test$Pred_result2 <- predict(plm_fit, newdata = test, type = "response")
submit_plm <- test[,.(Pred_result2)]
submit_plm[is.na(submit_plm$Pred_result2)] <- mean(train$SalePrice)
fwrite(test[,.(Id, result = submit_plm$Pred_result2)], "./project/volume/data/processed/submit_plm.csv")
```

For the supervised learning model, most of the coding structure is the same. We first load in the raw data and integrate the data to create one large dataset. Dummy variables are then defined for the training and testing sets. Before we create the model using XGBoost, we must set the tuning parameters. We tested using different values for

each parameter to find the values that created the most accurate model.

```
hyper_parm_tune <- NULL

myparam <- list(  objective           = "reg:squarederror",
                  gamma               = 0.02,
                  booster             = "gbtree",
                  eval_metric         = "rmse",
                  eta                 = 0.1,
                  max_depth           = 5,
                  min_child_weight    = 1,
                  subsample           = 1.0,
                  colsample_bytree    = 1.0,
                  tree_method         = 'hist'
)
XGBfit <- xgb.cv( params = myparam,
                  nfold = 5,
                  nrounds = 100000,
                  missing = NA,
                  data = dtrain,
                  print_every_n = 1,
                  early_stopping_rounds = 25)
```

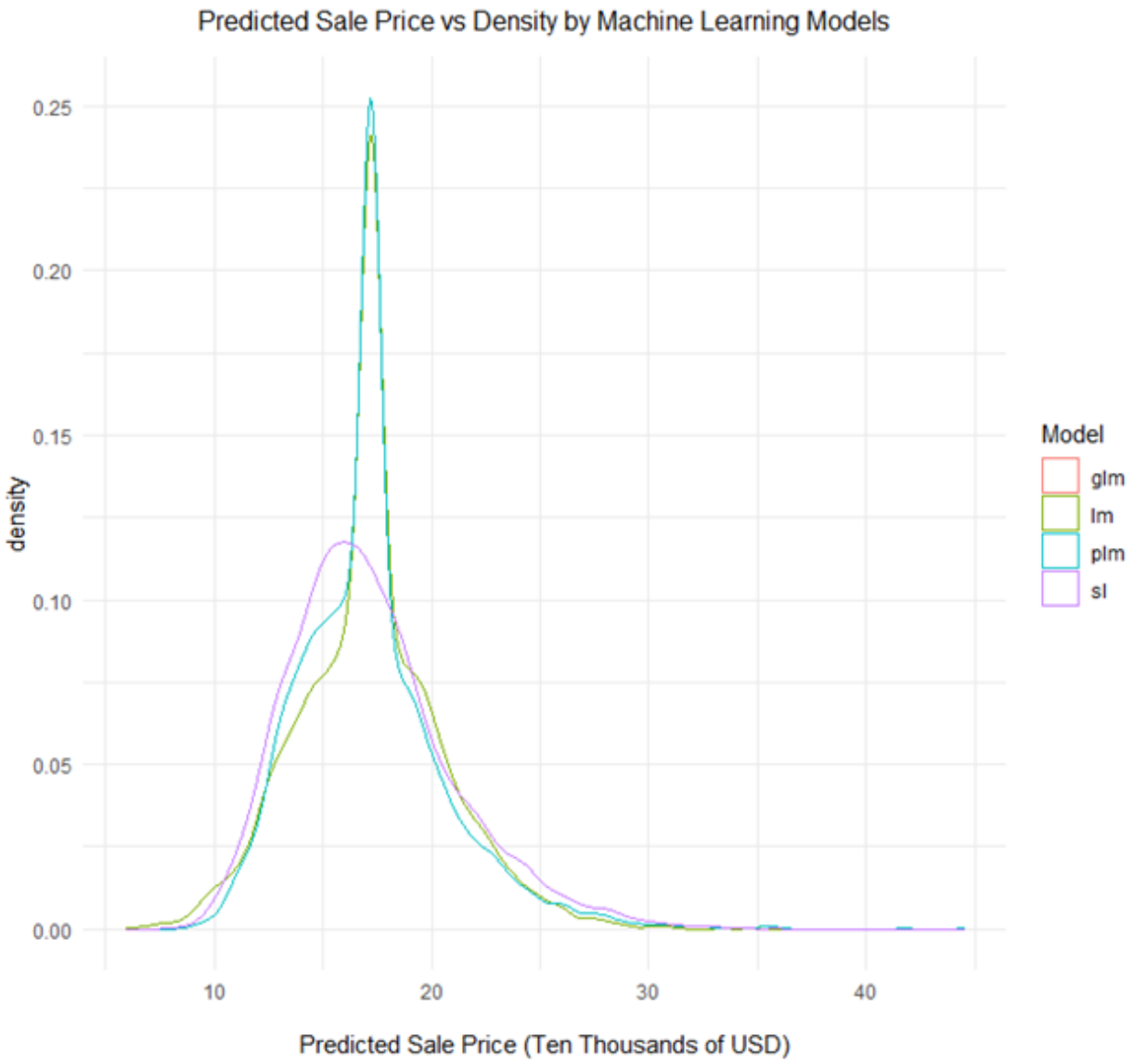
Finally we create and save the supervised learning model along with the dummy variables used, and predict the sales prices using said XGBoost model. Like we did with the regression models, we wrote the results onto a CSV file

```
best_tree_n <- unclass(XGBfit)$best_iteration
new_row <- data.table(t(myparam))
new_row$best_tree_n <- best_tree_n
test_error <- unclass(XGBfit)$evaluation_log[best_tree_n,]$test_rmse_mean
new_row$test_error <- test_error
hyper_parm_tune <- rbind(new_row, hyper_parm_tune)

watchlist <- list( train = dtrain)


XGBfit <- xgb.train( params = myparam,
                     nrounds = best_tree_n,
                     missing = NA,
                     data = dtrain,
                     watchlist = watchlist,
                     print_every_n = 1)


pred <- predict(XGBfit, newdata = dtest)

saveRDS(dummies,"./project/volume/models/SalePriceSL.dummies")
saveRDS(pred,"./project/volume/models/XGBfit.model")

test$Pred_SalePrice <- predict(XGBfit, newdata = dtest)
submit <- test[,.(Pred_SalePrice)]
submit[is.na(submit$Pred_SalePrice)] <- mean(train$SalePrice)

fwrite(test[,.(Id, SalePrice = submit$Pred_SalePrice)], "./project/volume/data/processed/submit_SL.csv")
```

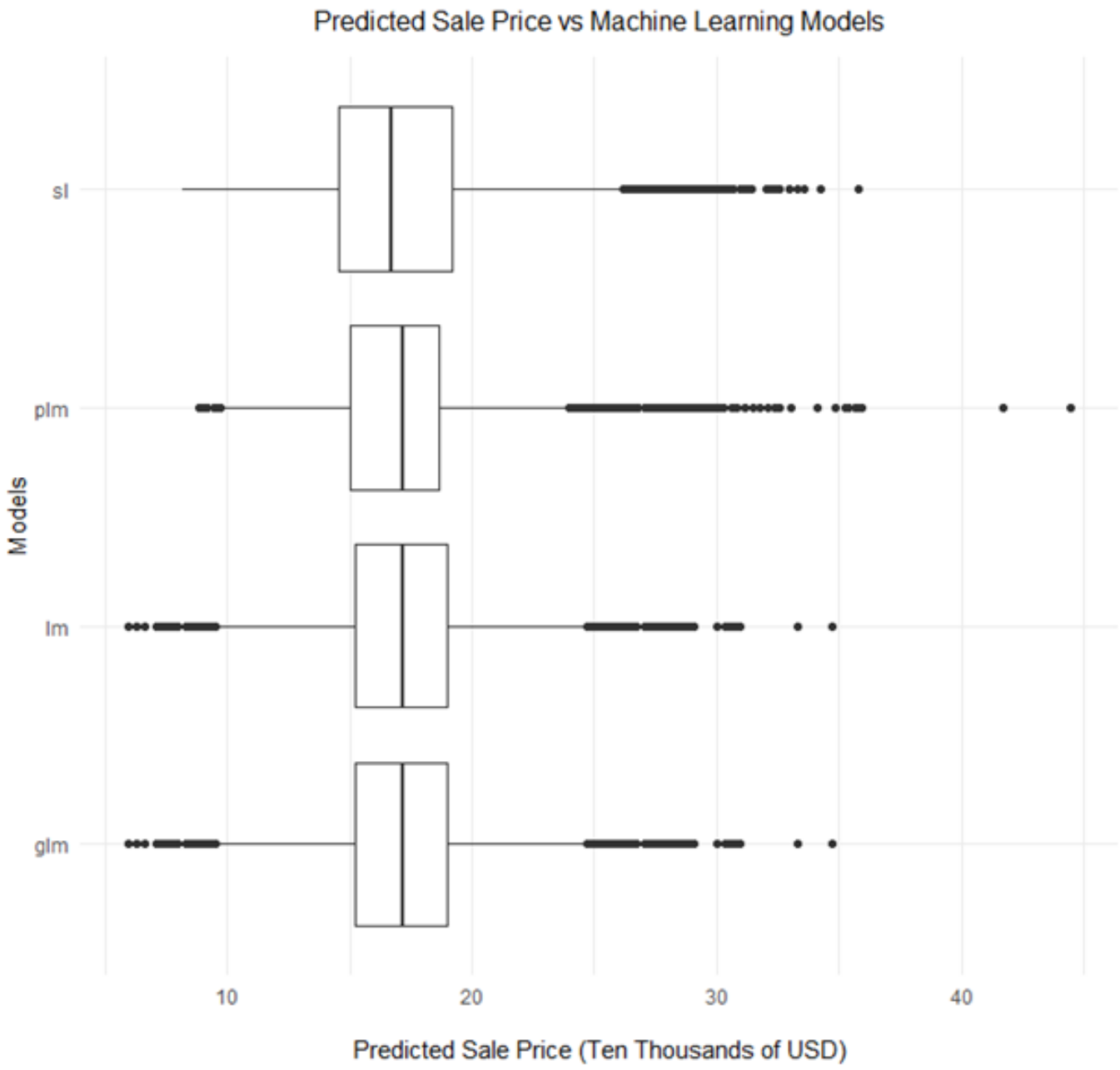
.

Results

The first graph listed is a density plot featuring all four models we used and how they compare to each other based on their Predicted Sale Price. The density plot shows the frequency of values per model and we want to see which values are present more often in each model. When you take a closer look at the graph, you only see three out of the four models present, and that's because Linear Regression (lm) and Gaussian Logistic Regression (glm) return the same values as each other. Because of this, we can assume the lm and glm models are no different than each other, but Poisson Logistic Regression (plm) returns a different plot than glm. Glm/lm are similar to plm in that they have a high density around $16,000-17,000 but are not exact. XGBoost (sl) follows a similar curve to the other three models, but doesn't go as high up in frequency up compared to the other models. The next graph listed is the four models compared against each other without any overlap. There is no change between values in the two graphs except this is a different view. The last graph is a boxplot featuring the Predicted Sale Price for each of the four models. Here we can see the maximum, minimum, IQR, median, and other sale prices predicted by the models. We can see the slight differences in medians, IQR, min, and max among the models. Plm has the highest predicted house sale prices which means there is a very large variation among the data.


Predicted Sale Price vs Density by Machine Learning Models
density
0.25
0.20
0.15
0.10
0.05
0.00
10
20
30
40
Predicted Sale Price (Ten Thousands of USD)
Model
glm
lm
plm
sl


Predicted Sale Price vs Density by Machine Learning Models
glm
lm
plm
sl
density
0.25
0.20
0.15
0.10
0.05
0.00
10
20
30
40
Predicted Sale Price (Ten Thousands of USD)
Model
glm
lm
plm
sl



In order to compare the four machine learning models we created, we decided to calculate the r-squared value, the root mean squared error (RMSE), and the mean absolute error (MAE). When comparing the linear and logistic regression models, we found that the linear regression model and the gaussian logistic regression model

performed exactly the same with an r-squared value of 0.6669, an RMSE value of 28346.32, and an MAE value of 21925.95. The poisson logistic regression model performed slightly better with an r-squared value of 0.6689, an RMSE value of 28127.63, and an MAE value of 21740.84. Once we created the supervised learning model using XGBoost however, we saw results that were significantly better than the regression models. The XGBoost model had an RMSE value of 21864.07 and an MAE value of 17354.22.

Conclusion

In conclusion, we found that the linear and logistic regression models performed well, but very similarly. The supervised learning model using XGBoost on the other hand performed significantly better with a 22.2% increase in performance based on RMSE score and a 20.2% increase in performance based on MAE score versus the highest performing logistic regression model (poisson).