HOUSING PRICE PREDICTION

CASE STUDY

Abstract

This project aims to predict housing prices using various features from a dataset, employing Ridge and Lasso regression techniques. Data cleaning, exploratory data analysis (EDA), and feature engineering steps were carried out to prepare the dataset. Ridge and Lasso models were optimized for hyperparameters, and their coefficients were analyzed to determine the most influential features. The project demonstrates the correlation between various factors like house size, garage area, and house prices.

Objective

The objective of this project is to develop a machine learning model that can accurately predict house prices based on various features using regression techniques such as Ridge and Lasso. It seeks to understand the impact of different house attributes on the price and determine the optimal feature set through feature selection techniques.

Introduction

Predicting house prices is a classic machine learning problem that has real-world applications in the real estate industry. The challenge involves analyzing numerous features like lot size, house condition, and neighborhood, and correlating them with the sale price. The dataset contains both categorical and

numerical features, which need to be carefully handled. In this project, we perform data cleaning, transformation, and exploratory data analysis, followed by the implementation of machine learning models to predict the sale price. Ridge and Lasso regression, both regularization techniques, are applied to minimize overfitting and to select important features.

Methodology

1) **Data Loading and Inspection:**

- The dataset is loaded and inspected for missing values, data types, and overall structure.
- Columns with more than 50% missing data are replaced with "None," and numerical columns are imputed using mean or mode.

2) **Data Transformation:**

- Logarithmic transformation is applied to the target variable, SalePrice, to handle skewness.
- Categorical variables are converted into dummy variables, and continuous variables are scaled for the machine learning models.

3) **Exploratory Data Analysis (EDA):**

- Boxplots and pie charts are used to visualize the distribution of numerical and categorical variables, respectively.
- Correlation matrices and heatmaps identify strong correlations between numerical variables.

4) **Feature Engineering:**

- New features, like the age of the property, are created.
- Feature selection techniques like Ridge and Lasso regression help to eliminate irrelevant features.

5) **Modeling and Evaluation:**

- Ridge and Lasso regressions are applied with hyperparameter tuning through GridSearchCV.

- Evaluation metrics such as R^2, RMSE are computed for both models to compare their performances.
- The coefficients of the models are analyzed to determine the most significant features influencing house prices.

Code

```
!pip install pytest-warnings

#import the warnings

import warnings

warnings.filterwarnings('ignore')


import numpy as np

import pandas as pd


pd.set_option('display.max_columns',None)

pd.set_option('display.max_rows',None)
```

Step 1: Import and Inspect Dataset

```
housing_data = pd.read_csv('train.csv')

housing_data.head()


housing_data.shape

housing_data.describe()

housing_data.info()

housing_data.isnull().sum()/housing_data.shape[0]*100
```

Step 2: Data Cleaning

```
cols =
["Alley","BsmtQual","BsmtCond","BsmtExposure","BsmtFinType1","BsmtFinType
2","FireplaceQu","GarageType","GarageFinish","GarageQual","GarageCond","Pool
QC","Fence","MiscFeature"]

for i in cols:
  housing_data[i].fillna("None",inplace = True)


housing_data.info()
```

Visualization

```
import matplotlib.pyplot as plt

import seaborn as sns


%matplotlib inline
```

 # Checking if the target variable SalePrice in normally distributed or any skewness

```
plt.figure(figsize=(6,6))

sns.distplot(housing_data['SalePrice'])

plt.show()
```

 # We can see that the target variable is right skewed.

```
print("Skewness: ",housing_data["SalePrice"].skew())

print("Kurtosis: ",housing_data["SalePrice"].kurtosis())
```

“We can observe that target variable has skewness greater than 1 and has high density around SalePrice of 16k.

Hence we can do data transformation (Here we're gonna do Logarithmic) for this variable”

Logrithmic Transformation

```
housing_data["SalePrice"] = np.log(housing_data["SalePrice"])


plt.figure(figsize=(6,6))
sns.distplot(housing_data['SalePrice'])
plt.show()


print("Skewness: ",housing_data["SalePrice"].skew())
print("Kurtosis: ",housing_data["SalePrice"].kurtosis())
```

#> Drop ID column

> Convert 'MSSubClass', 'OverallQual', 'OverallCond' to object DataType

> Convert 'LogFrontage', 'MasVnrArea' to numerical DataType

```
housing_data.drop("Id", axis=1, inplace=True)
housing_data[["MSSubClass","OverallQual","OverallCond"]] =
housing_data[["MSSubClass","OverallQual","OverallCond"]].astype(str)
housing_data["LotFrontage"] = pd.to_numeric(housing_data["LotFrontage"])
housing_data["MasVnrArea"] = pd.to_numeric(housing_data["MasVnrArea"])


housing_data.info()


null_cols = housing_data.columns[housing_data.isnull().any()]
```

```
null_cols


for i in null_cols:
    if housing_data[i].dtype == np.float64 or housing_data[i].dtype == np.int64:
        housing_data[i].fillna(housing_data[i].mean(), inplace=True)
    else:
        mode_value = housing_data[i].mode()
        if not mode_value.empty:  # Check if mode() returned a non-empty result
            housing_data[i].fillna(mode_value[0], inplace=True)
        else:
            # Handle case when there's no mode (e.g., all values are NaN)
            housing_data[i].fillna('Unknown', inplace=True)

housing_data.isna().sum()
```

Step 3: EDA on the Dataset

```
# List of Categorical Columns
cat_cols = housing_data.select_dtypes(include="object").columns
cat_cols


#List of Numerical Columns
num_cols = housing_data.select_dtypes(exclude='object').columns
num_cols
```

Univariant Analysis

```
# Numerical Columns
# Plotting Box Plot to visualize the distribution and check for any outliers
```

```
for i in num_cols:
  plt.figure(figsize=(8,6))
  sns.boxplot(housing_data[i])
  plt.show()
```

#From the Above box plot, We can see outliers in LotFrontage, LotArea, YearBulit, MasVnrArea, BsmtFinSF1,

#BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, 1stFlrSF, 2ndFlrSF, LowQualFinSF, GrLivArea, BsmtFullBath, BsmtHalfBath,

#BedroomAbvGr, KitchenAbrGr, TotRmAbvGrd, Fireplaces, GarageYrBlt, GarageCars, GarageArea, WoodDeckSF, OpenPorchSF,

#EnclosedPorch, 3SsnPorch, ScreenPorch, PoolArea, MiscVal, SalePrice

Numerical Columns

Plotting Box Plot to visualize the distribution

```
for i in cat_cols:
  plt.figure(figsize=(5,5))
  chart = housing_data[i].value_counts(normalize=True)
  print(chart)
  chart.plot.pie(labeldistance=None, autopct = '%1.2F%%')
  plt.legend()
  print("--------------------------------------------------------------------------------")
  plt.show()
```

>From the Above pie plot, We can look percentage values in categorical columns and we can infer that , "MS Zoning", 'Street', 'landContour', 'Utilities', 'LotConfiguration' , 'LandSlope'etc.....

> All these columns are having more than 70% of a distribution in a single category.

Bivariate/ Mulitvariate Analysis on the dataset

Plot of MSZoning v/s LotFrontage

```
sns.barplot(x='MSZoning',y='LotFrontage',data = housing_data)
plt.show()
```

Plot of MSSubClass v/s LotFrontage

```
plt.figure(figsize=(10,8))
sns.barplot(x='MSSubClass',y='LotFrontage',data = housing_data)
plt.show()
```

Plot of HouseStyle v/s SalePrice based on Street

```
sns.barplot(x='HouseStyle',y='SalePrice',hue='Street' ,data = housing_data)
plt.show()
```

Plot of BldgType v/s SalePrice

```
sns.barplot(x='BldgType', y='SalePrice', hue='Street', data= housing_data)
plt.show()
```

Plot of BsmtQual v/s SalePrice

```
sns.barplot(x='BsmtQual', y='SalePrice',data= housing_data)
plt.show()
```

Conclusion :

> We can see that RL(Residential Low Density) has the highest LotFrontage and RM (Residential Medium Density) has the least.

> We can see that 2-STORY 1946 and NEWER has the highest LotFrontage and PUD - MULTILEVEL - INCL SPLIT LEV/FOYER has the least.

> The SalePrice is not showing much variance with respect to the Style of Dwelling (one story/ two story)

> The SalePrice is almost same for all the Building Types(Types of Dwelling) and the Basement Quality, so there is no significant pattern.

Calculating Age of the property

```
housing_data['Age'] = housing_data["YrSold"] - housing_data["YearBuilt"]

housing_data["Age"].head()


housing_data.drop(["YearBuilt","YrSold"], axis=1, inplace=True)

housing_data.head()
```

Correlation between Numerical Columns

```
plt.figure(figsize=[25,25])

sns.heatmap(housing_data.corr(numeric_only = True),annot=True,cmap
='BuPu')

plt.title("Correlation Between Numerical Columns")

plt.show()


housing_data.corr(numeric_only = True)
```

>Get top 10 correlated columns

```
k = 10

plt.figure(figsize=[10,10])
```

```
cols = housing_data.corr(numeric_only = True).nlargest(k,'SalePrice').index
cm = np.corrcoef(housing_data[cols].values.T)
sns.heatmap(cm,annot=True, square = True, fmt = '.2f', cbar = True, annot_kws = {'size':10},
        yticklabels = cols.values, xticklabels = cols.values)
plt.show()
```

Conclusion

We can see that -

> GarageArea and GarageCars are highly correlated with coff of 0.88

> GrLiveArea and TotRmsAbvGrd are highly correlated with coff of 0.83

> TotalBsmtSF and 1stFlrSF are highly correlated with coff of 0.82

#PairPlot for Numerical Columns

```
cols = ["SalePrice", "OverallQual", "GrLivArea",
"GarageCars","TotalBsmtSF","FullBath" ,"Age"]
plt.figure(figsize=[10,7])
sns.pairplot(housing_data[cols])
plt.show()
```

 # We can see that Age has negative correlation with target variable SalePrice and TotalBSmtSF and GrLivArea have postive correlation with SalePrice

Step 4: Data Preparation

#Dummy Encoding

```
housing_num = housing_data.select_dtypes(include= ['int64','float64'])
```

```
housing_cat = housing_data.select_dtypes(include= ['object'])
housing_cat

housing_cat_dm = pd.get_dummies(housing_cat,drop_first = True, dtype= int)
housing_cat_dm

house = pd.concat([housing_num,housing_cat_dm],axis = 1)
house.head()

# Splitting up the dataset
# First let's create the dependent and independent variable

#### Split into Target and Feature

X = house.drop(['SalePrice'],axis=1).copy()
y = house['SalePrice'].copy()
X.head()
y.head()

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state =
42)
X_train

# Scaling the Dataset
```

```
num_cols = list(X_train.select_dtypes(exclude='object').columns)

scaler = StandardScaler()

X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])

# Creating a function to calculate  evaluation metrices
def evaluate_metrices(y_train,y_train_pred,y_test,y_pred):

  # r^2 values for train and test data
  print("r2 score (train) = ",'%2.2f' % r2_score(y_train,y_train_pred))
  print("r2 score (test) = ",'%2.2f' % r2_score(y_test,y_pred))

  ## RMSE for train and test data
  mse_train = mean_squared_error(y_train,y_train_pred)
  mse_test = mean_squared_error(y_test,y_pred)

  rmse_train = np.sqrt(mse_train)
  rmse_test = np.sqrt(mse_test)

  print("RMSE(Train) = ", '%2.2f' % rmse_train)
  print("RMSE(Test) = ", '%2.2f' % rmse_test)
```

Step 5: Build ML Model

Import ML Libs

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.linear_model import Ridge, Lasso
from sklearn.model_selection import GridSearchCV
```

Applying Ridge Regression with varying the hyperparameter 'Lambda'

```
params = { 'alpha':
        [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5,
       0.6, 0.7, 0.8, 0.9, 1.0,2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10, 20, 50, 100,
500, 1000]}
ridge = Ridge()
ridgeCV = GridSearchCV(estimator = ridge, param_grid = params, scoring =
'neg_mean_absolute_error', cv=5,
                    return_train_score = True, verbose =1, n_jobs = -1)
ridgeCV.fit(X_train,y_train)

ridgeCV.best_params_
ridgeCV.cv_results_
ridge = Ridge(alpha = 100)
ridge.fit(X_train,y_train)
ridge.coef_

y_train_pred = ridge.predict(X_train)
y_pred = ridge.predict(X_test)
```

```
evaluate_metrices(y_train, y_train_pred, y_test, y_pred)


ridgeCV_result = pd.DataFrame(ridgeCV.cv_results_)
ridgeCV_result


plt.plot(ridgeCV_result['param_alpha'], ridgeCV_result['mean_train_score'], label
= 'train')
plt.plot(ridgeCV_result['param_alpha'], ridgeCV_result['mean_test_score'], label =
'test')
plt.xlabel('alpha')
plt.ylabel('R2_score')
plt.xscale('log')
plt.title('Ridge Regression')
plt.legend()
plt.show()


# Applying Lasso Regression with varying the hyperparameter 'Lambda'
lasso = Lasso()
lassoCV = GridSearchCV(estimator = lasso, param_grid = params, scoring =
'neg_mean_absolute_error', cv=5,
                       return_train_score = True, verbose =1, n_jobs = -1)
lassoCV.fit(X_train,y_train)


lassoCV.best_params_
lasso = Lasso(alpha = 0.001)
lasso.fit(X_train, y_train)


lasso.coef_
```

```
y_train_pred1 = lasso.predict(X_train)
y_pred1 = lasso.predict(X_test)

evaluate_metrices(y_train, y_train_pred1, y_test, y_pred1)

lassoCV_result = pd.DataFrame(lassoCV.cv_results_)
lassoCV_result

plt.plot(lassoCV_result['param_alpha'], lassoCV_result['mean_train_score'], label = 'train')
plt.plot(lassoCV_result['param_alpha'], lassoCV_result['mean_test_score'], label = 'test')
plt.xlabel('alpha')
plt.ylabel('R2_score')
plt.xscale('log')
plt.title('Lasso Regression')
plt.legend()
plt.show()

## Feature Extraction/ Elimination
betas = pd.DataFrame(index = X_train.columns)
betas.rows = X.columns

# Creating columns for Ridge and Lasso Coefficient against each feature
betas['Ridge'] = ridge.coef_
betas['Lasso'] = lasso.coef_
betas
```

#View the features removed by lasso

```
lasso_cols_removed = list(betas[betas['Lasso']==0].index)
print(lasso_cols_removed)
```

#View the features selected by lasso

```
lasso_cols_selected = list(betas[betas['Lasso']!=0].index)
print(lasso_cols_selected)
```

```
print('Number of removed Lasso columns: ',len(lasso_cols_removed))
print('Number of selected Lasso columns: ',len(lasso_cols_selected))
```

View the top 10 coefficient of Ridge regression in descending order

```
betas['Ridge'].sort_values(ascending=False)[:10]
```

We have to take inverse log of betas to interpret the ridge coefficents in terms of target variable

```
ridge_coeffs = np.exp(betas['Ridge'])
ridge_coeffs.sort_values(ascending = False)[:10]
```

View the top 10 coefficient of Lasso regression in descending order

```
betas['Lasso'].sort_values(ascending=False)[:10]
```

We have to take inverse log of betas to interpret the lasso coefficents in terms of target variable

```
lasso_coeffs = np.exp(betas['Lasso'])
lasso_coeffs.sort_values(ascending = False)[:10]
```

Conclusion:

 # Below are the Top 10 features with corresponding coefficients according to Ridge model

- GrLivArea 1.045704
- 1stFlrSF 1.030758
- OverallQual_9 1.028232
- 2ndFlrSF 1.026525
- RoofMatl_Tar&Grv 1.025966
- OverallQual_8 1.025820
- GarageCars 1.025528
- PoolArea 1.024453
- RoofMatl_CompShg 1.022923
- Neighborhood_NridgHt 1.022469

 # Below are the Top 10 features with corresponding coefficients according to Lasso model

- PoolArea 1.164807
- PoolQC_None 1.144479
- GrLivArea 1.117957
- OverallQual_9 1.032962
- OverallQual_8 1.032376
- RoofMatl_Tar&Grv 1.030344
- RoofMatl_CompShg 1.029419

- TotalBsmtSF 1.026382

- Neighborhood_Crawfor 1.023844

- GarageCars 1.023315

Few Inferences are:

- Therefore the price of the house will increase by 1.11 with the increase in GrLivArea

- The price of house can increase by 1.02 times if the finish of the house is very good.

- If the house has GarageCars the price may increase by 1.02 times

- If the house has Pool Area then the price can increase by 1.16 times.

- If the basement condition is typical then the house price may increase upto 1.03

The optimal value of lambda for Ridge Regression is 100

The optimal value of lambda for Lasso Regression is 0.001

Conclusion

- **Ridge and Lasso Regression Findings:**
    - Ridge Regression identified top features like GrLivArea, 1stFlrSF, OverallQual_9, and GarageCars.
    - Lasso Regression found features like PoolArea, GrLivArea, and OverallQual_9 as the most significant.
    - Lasso removed irrelevant features, while Ridge focused on regularizing all the coefficients.
- **Inferences:**
    - Large living areas (GrLivArea) and high-quality finishes (OverallQual) significantly impact house prices.

    - The presence of a pool and a garage also positively influence the price.

- Optimal Lambda:
    - The optimal values of lambda for Ridge and Lasso were 100 and 0.001, respectively, indicating their regularization strength.

This project highlights how effective regression techniques can be in identifying key factors affecting housing prices and offers insights for further improvement by applying different models or incorporating more advanced feature selection techniques.