House Price Prediction using Advance Regression Techniques

Amit Khandelwal, Achala Bhati, Shyamli Rao

April 29, 2017

Contents

Chapter 1

Introduction

Real estate industry is one of the most important industry in the world Jae Kwon Bae (2015) Limsombunchao, V. (2004). It impacts largely on the economic development of the country and people's fundamental needs. So, accurately predicting the real estate price (or house price prediction) is of great importance Dubin, Robin A. (1998).

1.1 Problem Statement

We will use advanced regression techniques such as SVR (support vector regression), neural networks, ridge regression etc. to predict the sales price of a house. We are using a dataset with 79 feature variables describing every aspect of houses in the town of Ames, Iowa. Our to-do list is as follows:

1. Implement at least 3 different regression techniques.
2. Explain why one performs better than the other
3. Critically analyze the paper's findings.

1.2 Approach

First we will try to understand the data and try to figure out relevant features out of it. We will also explore if any form of normalization or data preprocessing is needed. Once this is done we will focus on feature extraction and selection. Finally, we will fit several Machine Learning models and will evaluate their results. We will also find out why a particular model performs better than other. A comparative study will help us get more insight into which model to use with which kind of data.

Chapter 2

Literature Survey

2.1 Using SVM with PSO

In the paper entitled *Real estate price forecasting based on SVM optimized by PSO* Yubiao Wang (2013) , they use SVM regression technique to predict the house price. The parameters of SVM equation are found by the particle swarm algorithm (PSO). They have used kernelized version of SVR. In the end try to quickly find reasonable accurate SVM parameters. These are steps followed by them :

1. Normalize data : As a first step after feature extraction they normalize the data using this formula : $x_{ik} = \frac{x_{ik} - x_k^{min}}{x_k^{max} - x_k^{min}}$

2. Apply PSO algorithm : Each particle has position vector $X_i = (x_{i1}, x_{i2}, ..., x_{iD})$ and velocity vector $V_i = (v_{i1}, v_{i2}, ..., v_{iD})$, i = 1,2, ..., n. Each particle is represented as a potential solution to a problem in D-dimensional search space. After every iteration, each particle is accelerated towards the particles previous best position $P_i = (p_{i1}, p_{i2}, ..., p_{iD})$ and global best position $P_g = (p_{g1}, p_{g2}, ..., p_{gD})$.
   The updates of velocity and position are as follows :
   $v_{id}^{l+1} = w * v_{id}^l + c_1 * rd_1^l * (p_{id}^l - x_{id}^l) + c_2 * rd_2^l * (p_{gd}^l - x_{id}^l)$
   $x_{id}^{l+1} = v_{id}^{l+1} + x_{id}^l$
   where,
   w : inertial weight coefficient
   c1,c2 : learning factor or acceleration coefficient
   rd_1^l, rd_2^l : random positive number
   l : iteration

3. w controls the impact of previous history of velocities to current velocity. As w increases global exploration is more, as w decreases local exploration is more. Hence to balance out this they have kept w a linearly decreasing function.
   $w(k) = w_{start} - k * \frac{w_{start} - w_{end}}{T_{max}}$
   where,
   k : current iteration
   T_{max} : maximum iterations

4. Calculate MAPE (mean absolute percentage error) for each particle and get $P_i and P_g$. Update $V_i and X_i$ and search for better SVM parameters.

5. As number of iterations become greater than T_{max} or the MAPE becomes less than threshold, the PSO algorithm stops and returns the P_g, the optimum parameters of SVM.

2.2 Data Set

The data set which we are using has 1461 data points and 79 features, which are as follows :

- SalePrice - the property's sale price in dollars. This is the target variable that you're trying to predict.
- MSSubClass: The building class
- MSZoning: The general zoning classification
- LotFrontage: Linear feet of street connected to property
- LotArea: Lot size in square feet
- Street: Type of road access
- Alley: Type of alley access
- LotShape: General shape of property
- LandContour: Flatness of the property
- Utilities: Type of utilities available
- LotConfig: Lot configuration
- LandSlope: Slope of property
- Neighborhood: Physical locations within Ames city limits
- Condition1: Proximity to main road or railroad
- Condition2: Proximity to main road or railroad (if a second is present)
- BldgType: Type of dwelling
- HouseStyle: Style of dwelling
- OverallQual: Overall material and finish quality
- OverallCond: Overall condition rating
- YearBuilt: Original construction date
- YearRemodAdd: Remodel date
- RoofStyle: Type of roof
- RoofMatl: Roof material

- Exterior1st: Exterior covering on house
- Exterior2nd: Exterior covering on house (if more than one material)
- MasVnrType: Masonry veneer type
- MasVnrArea: Masonry veneer area in square feet
- ExterQual: Exterior material quality
- ExterCond: Present condition of the material on the exterior
- Foundation: Type of foundation
- BsmtQual: Height of the basement
- BsmtCond: General condition of the basement
- BsmtExposure: Walkout or garden level basement walls
- BsmtFinType1: Quality of basement finished area
- BsmtFinSF1: Type 1 finished square feet
- BsmtFinType2: Quality of second finished area (if present)
- BsmtFinSF2: Type 2 finished square feet
- BsmtUnfSF: Unfinished square feet of basement area
- TotalBsmtSF: Total square feet of basement area
- Heating: Type of heating
- HeatingQC: Heating quality and condition
- CentralAir: Central air conditioning
- Electrical: Electrical system
- 1stFlrSF: First Floor square feet
- 2ndFlrSF: Second floor square feet
- LowQualFinSF: Low quality finished square feet (all floors)
- GrLivArea: Above grade (ground) living area square feet
- BsmtFullBath: Basement full bathrooms
- BsmtHalfBath: Basement half bathrooms
- FullBath: Full bathrooms above grade
- HalfBath: Half baths above grade
- Bedroom: Number of bedrooms above basement level

- Kitchen: Number of kitchens
- KitchenQual: Kitchen quality
- TotRmsAbvGrd: Total rooms above grade (does not include bathrooms)
- Functional: Home functionality rating
- Fireplaces: Number of fireplaces
- FireplaceQu: Fireplace quality
- GarageType: Garage location
- GarageYrBlt: Year garage was built
- GarageFinish: Interior finish of the garage
- GarageCars: Size of garage in car capacity
- GarageArea: Size of garage in square feet
- GarageQual: Garage quality
- GarageCond: Garage condition
- PavedDrive: Paved driveway
- WoodDeckSF: Wood deck area in square feet
- OpenPorchSF: Open porch area in square feet
- EnclosedPorch: Enclosed porch area in square feet
- 3SsnPorch: Three season porch area in square feet
- ScreenPorch: Screen porch area in square feet
- PoolArea: Pool area in square feet
- PoolQC: Pool quality
- Fence: Fence quality
- MiscFeature: Miscellaneous feature not covered in other categories
- MiscVal: Value of miscellaneous feature
- MoSold: Month Sold
- YrSold: Year Sold
- SaleType: Type of sale
- SaleCondition: Condition of sale

Chapter 3

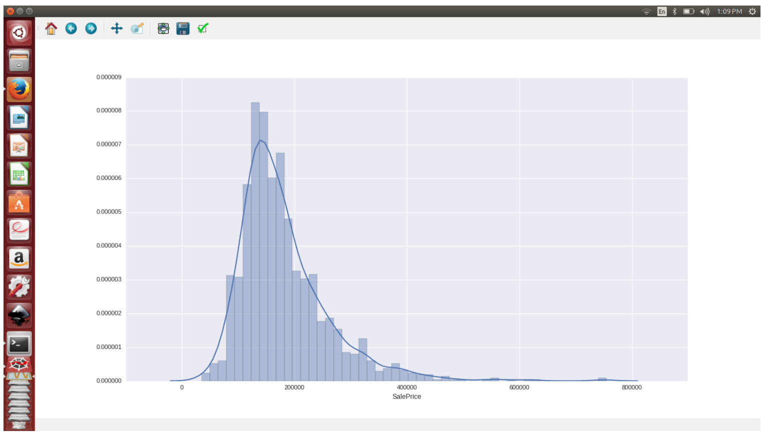
Approach

3.1 Data Analysis

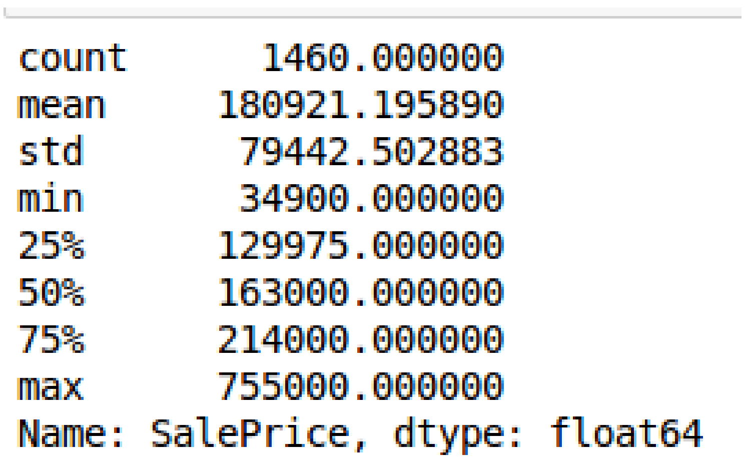
This section includes the train data analysis for house sales price prediction. We have analyzed the output parameter i.e. the sales price of house and the features which are highly correlated. Also we categorized the input features, classifying them whether its categorical or numeric. And we rated whether the feature is of high, medium or low importance.

3.1.1 Sales price

We analyzed the sales prices parameter in the train data. From the graph below we can see that sales price is positively skewed. The skewness of the data represents tells whether the deviations from the mean would be positive or negative. Since our data is positively skewed most the deviations from the mean would be positive.



When we describe the feature, sales price then following data comes up. The mean, standard deviation, minimum and maximum are shown.

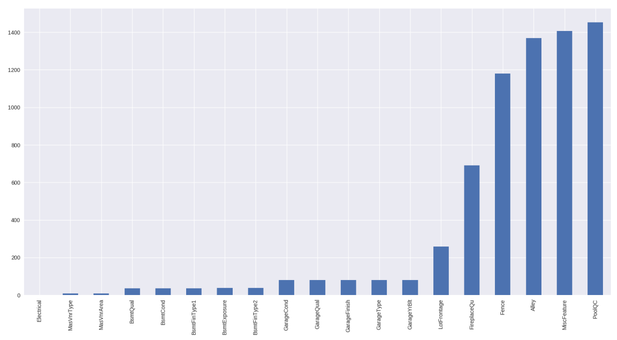


```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

3.1.2 Data Preprocessing

Removing Missing Values

this graph shows the missing value distribution of data. We replaced these missing values



with the mean of the feature.

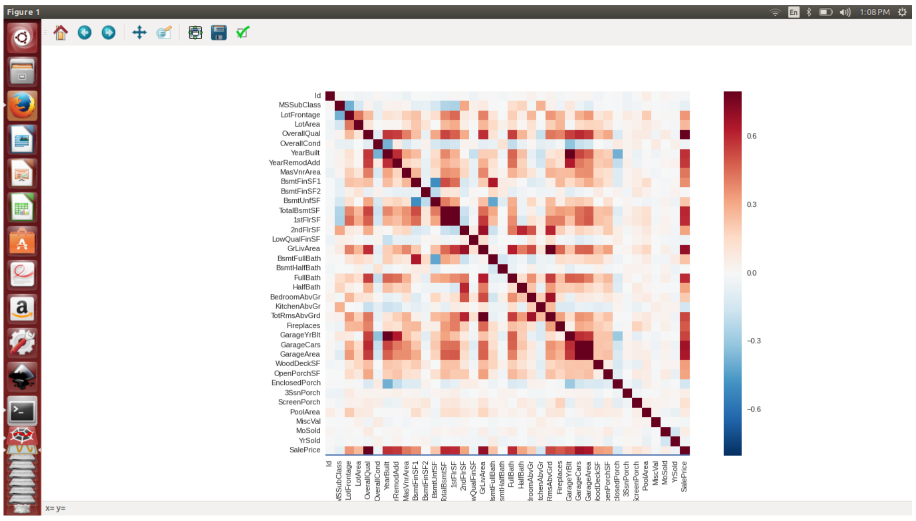
Encoding categorical data

We encoded categorical data by mapping the various categories to integer values using *LabelEncoder* library.

3.1.3 Feature selection

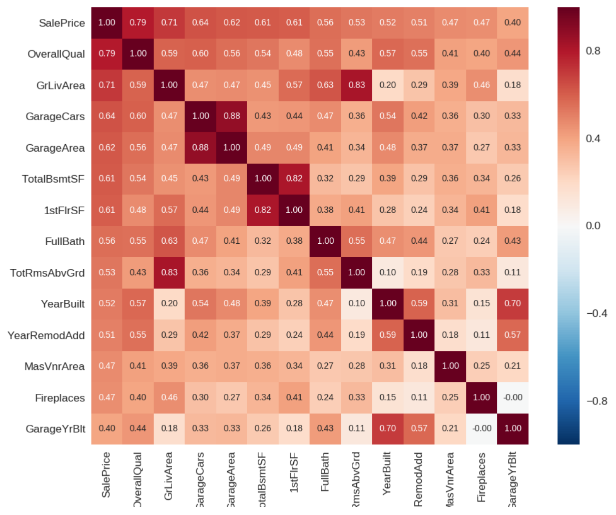
We would like to see which features affect the most to the sale price, hence we build a correlation matrix. From the figure below, where red means highly correlated features, we observed that these 13 features highly affect the house sales price.

- OverallQual
- YearBuild
- FullBath
- Fireplaces



- YearRemodAdd
- MarVnrArea
- TotalBsmtSF
- 1stFlrSF
- GrLivArea
- TotRmsAbvGrd
- GarageYrBld
- GarageArea
- GarageCars

Using these 13 features we created another correlation matrix from which we selected 9 features. These features are as follows :

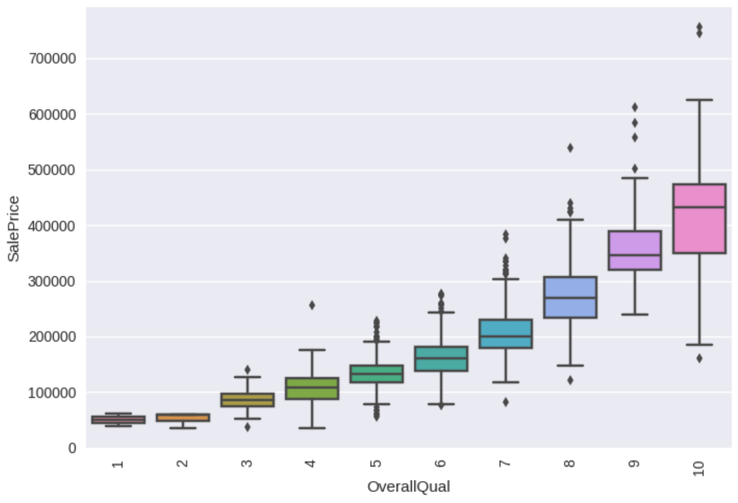


- OverallQual
- YearBuild
- FullBath
- Fireplaces
- YearRemodAdd
- MasVnrArea
- TotalBsmtSF
- GrLivArea
- GarageArea

3.1.4 Feature graphs

For each of the above feature selected we analyzed their behavior with Sales price using graphs.

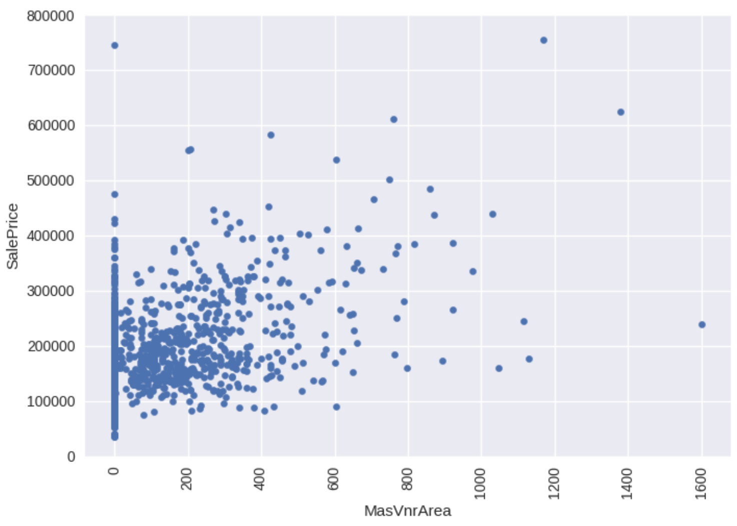
OverallQual



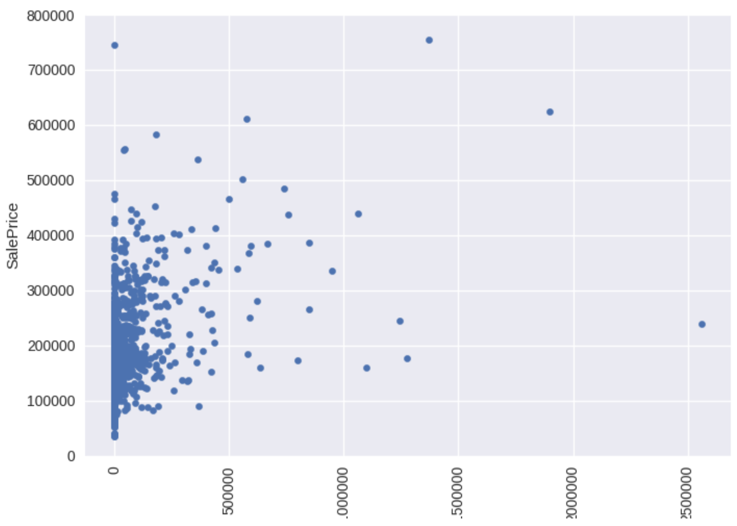
As shown in graph we can see that the Sales price is quadratically increasing as the overall Quality is increasing. This also matches our prediction that the sales price must increase as the quality of house is improving. Hence we have applied quadratic function to this feature.

MasVnrArea

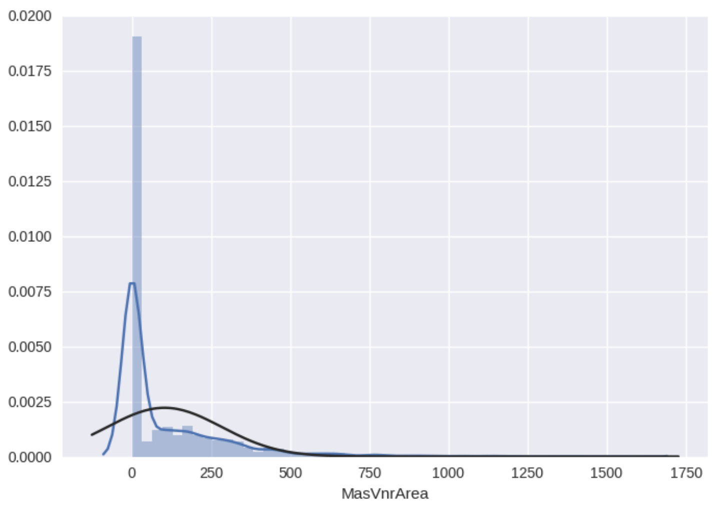
As we can see from the figure the data points are skewed near zero, so we have applied



Below is the relationship of this feature when quadratic function is applied. As we can see the plot is now linear. Below is the distribution of this feature which shows that it is

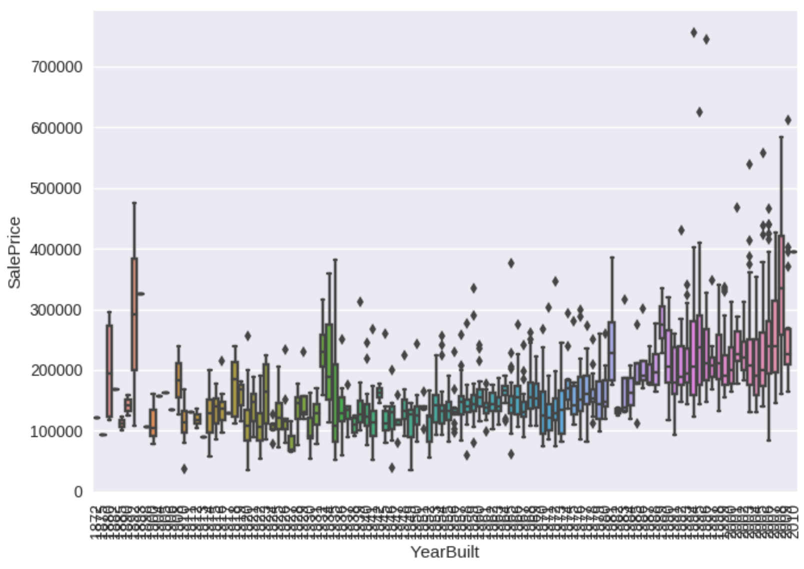


positively skewed. So by taking log it has become normally distributed.



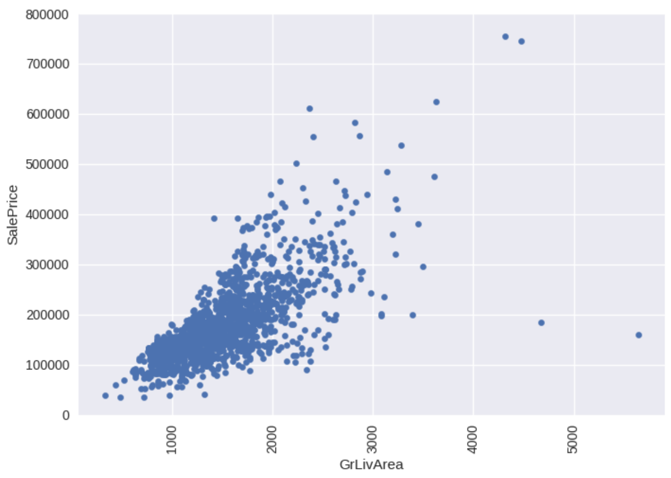
YearBuild

As shown in graph we can see that the Sales price is quadratically increasing as the year built is increasing. This also matches our prediction that the sales price must increase as the age of house is decreasing. Hence we have applied quadratic function to this feature.

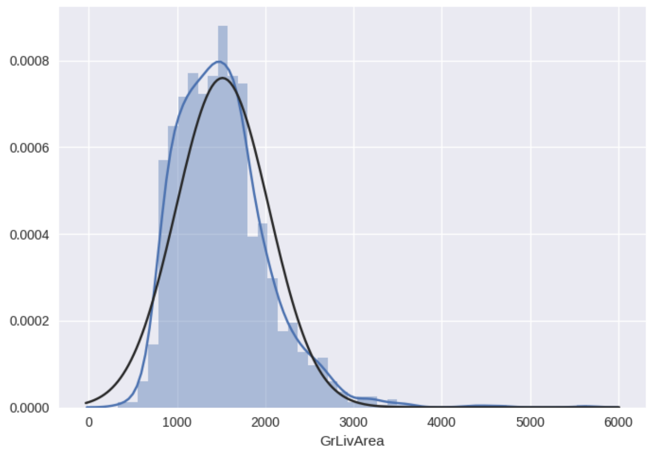


GrLivArea

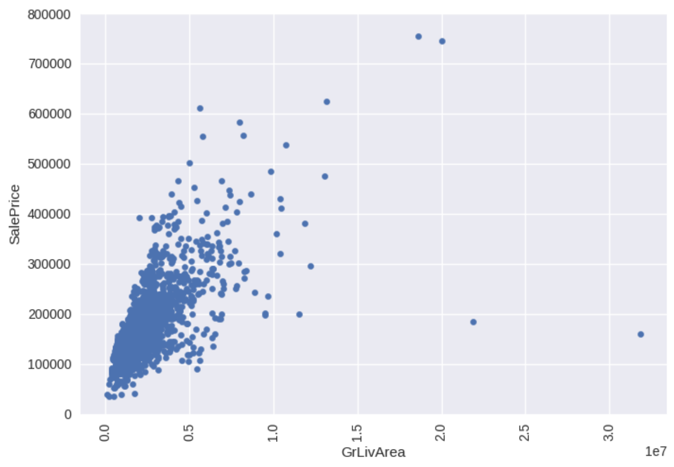
This is the actual plot of data points with respect to Sales prices. Below is the distribution



of this feature which shows that it is positively skewed. So by taking log it has become normally distributed.

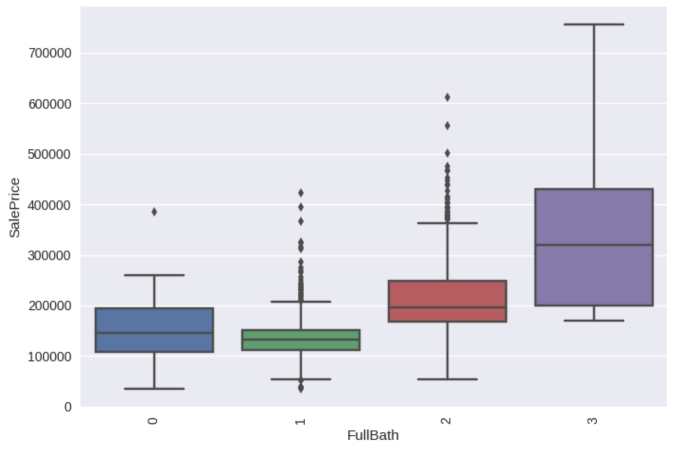


As shown in graph we can see that the GrLivArea is quadratically increasing as Living Area is increasing. Hence we have applied quadratic function to this feature.



FullBath

As shown in graph we can see that the Sales price is quadratically increasing as the FullBath is increasing. This also matches our prediction that the sales price must increase as the value of FullBath increase. Hence we have applied quadratic function to this feature.



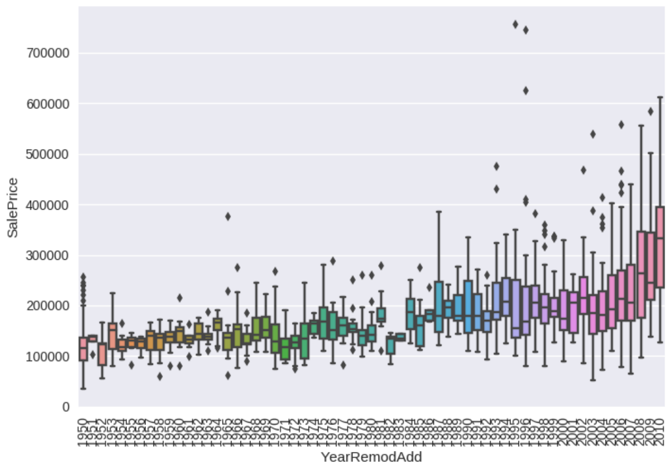
YearRemodAdd

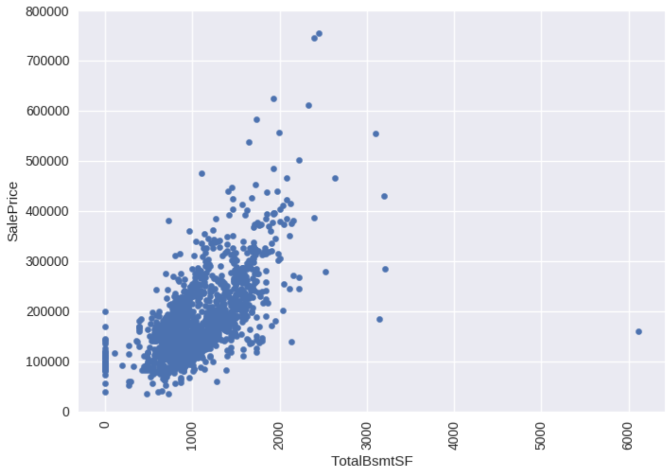
As shown in graph we can see that the Sales price is quadratically increasing as the year built is increasing. This also matches our prediction that the sales price must increase as the age of house is decreasing. Hence we have applied quadratic function to this feature.

TotalBsmtSF

This is the actual plot of data points wrt Sales prices.

As shown in graph we can see that the SalePrice is quadratically increasing as the basement area is increasing. Hence we have applied quadratic function to this feature.

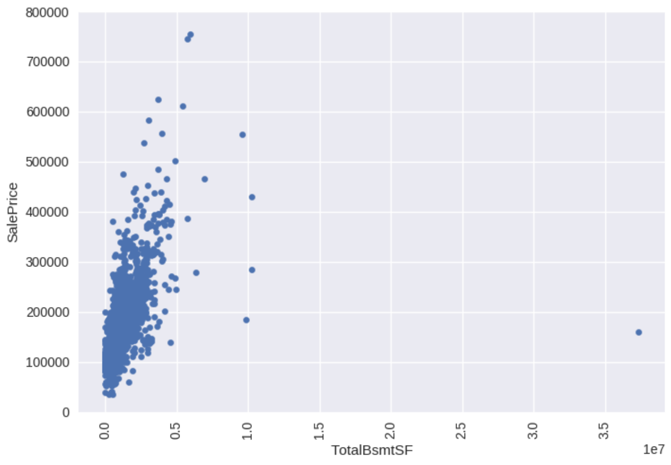


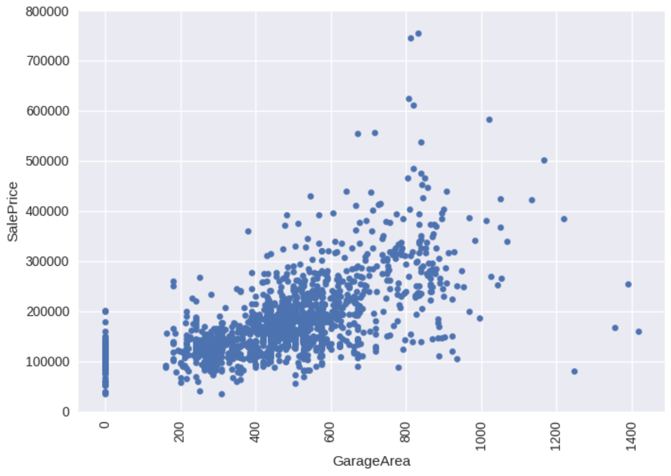


GarageArea

This is the actual plot of data points wrt Sales prices.

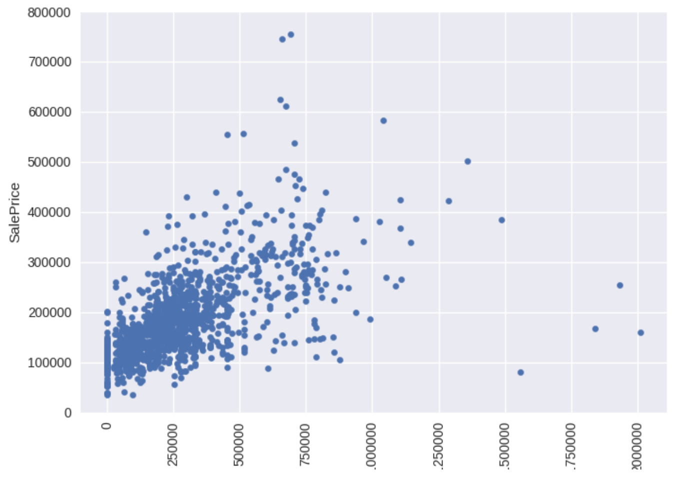
As shown in graph we can see that the Sales Price is quadratically increasing as the Garage area is increasing. Hence we have applied quadratic function to this feature.

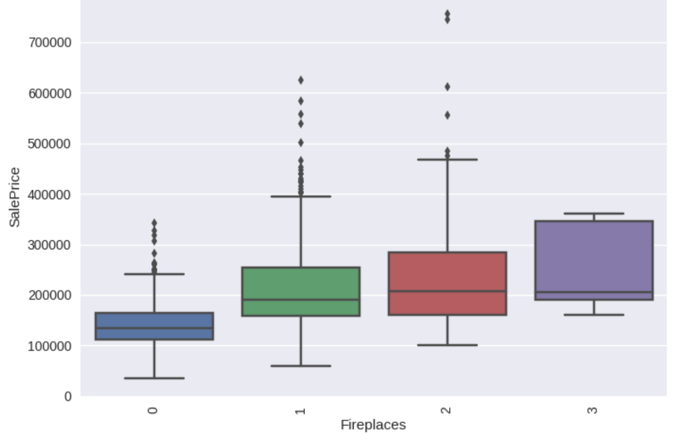




Fireplaces

As shown in graph we can see that the Sales price is quadratically increasing as the Fireplaces is increasing. This also matches our prediction that the sales price must increase as the value of Fireplaces increase. Hence we have applied quadratic function to this feature.





3.2 Implementation

We applied ridge regression, Kernelized Ridge Regression, Lasso Regression, Feed Forward Neural Network, Support Vector Machines, Partial Least Square, Stochastic gradient decent, Bayesian regression and Boosting to the train data set considering all the features and also by doing feature engineering. SalePrice's log tansformation has been taken so that it becomes more "normal". Therefore we can observe that the skewness of data has decreased. The python code is as follows :

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Apr 26 15:05:55 2017

@author: achala
"""
#import xgboost as xgb
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from scipy import stats
import warnings
from sklearn.linear_model import Ridge,Lasso
from sklearn.cross_validation import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder

from sklearn.neural_network import MLPRegressor

from sklearn.kernel_ridge import KernelRidge
from sklearn import svm

from sklearn.linear_model import SGDRegressor

from sklearn import linear_model

from sklearn.cross_decomposition import PLSRegression

from sklearn.cross_validation import KFold
from sklearn.cross_validation import train_test_split




raw_train = pd.read_csv('processed_train.csv')
raw_test = pd.read_csv('processed_test.csv')
#print (raw_train.head)
raw_train = raw_train.replace(to_replace='-inf',value=np.nan)
raw_train = raw_train.replace(to_replace='inf',value=np.nan)

raw_test = raw_test.replace(to_replace='-inf',value=np.nan)
raw_test = raw_test.replace(to_replace='inf',value=np.nan)


raw_train = raw_train.apply(lambda x:x.fillna(x.mean()))


raw_test = raw_test.apply(lambda x:x.fillna(x.mean()))

#print (raw_train)
'''
cat_list = []
for i in raw_train.columns:
    if raw_train[i].dtype == object:
        raw_train[i] = raw_train[i].astype('category')
        cat_list += [i]
```

```
#print (cat_list)

number = LabelEncoder()
for i in cat_list:
    raw_train[i]=number.fit_transform(raw_train[i].astype('str'))

cat_list = []
for i in raw_test.columns:
    if raw_test[i].dtype == object:
        raw_test[i] = raw_test[i].astype('category')
        cat_list += [i]




#print (cat_list)

number = LabelEncoder()
for i in cat_list:
    raw_test[i]=number.fit_transform(raw_test[i].astype('str'))

corelation_mat  = raw_train.corr()
sns.heatmap(corelation_mat, vmax=.8, square=True);
'''
X_train= raw_train.loc[:,'OverallQual':'YearRemodAdd'];

X_test=raw_test.loc[:,'OverallQual':'YearRemodAdd'];

#raw_train['SalePrice'].describe()


Y_train=raw_train.SalePrice


################################################## Ridge ##############################
## 0.9316
X1 = np.array(X_train)
Y1 = np.array(Y_train)
X_test1 = np.array(X_test)
reg = Ridge(alpha = 0.1)
reg.fit(X1,Y1);
op=reg.predict(X_test1);


yo1 = pd.DataFrame(columns=["Id","SalePrice"])
yo1["Id"]=raw_test.Id.values
yo1["SalePrice"]=op
```

```
yo1.to_csv('predictions_ridge.csv',index=False)
print (op);

################################################# Ridge ##############################
############################################### Ridge cross validation ##########
######## 0.55
def model_generation(X_train,y_train,model):
    kf=KFold(X_train.shape[0], n_folds=8)
    rf = Ridge(alpha = 0.1)
    if(model=="Lasso"):
        rf = Lasso(alpha = 0.1)
    if(model=="kernel"):
        rf = KernelRidge(alpha=1, kernel='linear', gamma=None, degree=3, co
    if(model=="Neural"):
        rf = MLPRegressor(hidden_layer_sizes=(100, ), activation='relu',lea
    if(model=="svm"):
        rf = svm.SVR()
    if(model=="partial"):
        rf = PLSRegression()
    if(model=="Bay"):
        rf = linear_model.BayesianRidge()
    err=list()
    min_rf = rf
    min_e=100
    for a,b in kf:
        print ("hrrrrr")
        X_trn, X_tst = X_train[a] , X_train[b]
        y_trn, y_tst = y_train[a], y_train[b]
        rf.fit(X_trn,y_trn)
        y_pred=rf.predict(X_tst)
        err = mean_squared_error(y_tst, y_pred)
        if(err < min_e):
            min_e = err
            min_rf = rf
    return min_rf


X_train, X_test, y_train, y_test = train_test_split(X1, Y1, test_size=0.33,
rf=model_generation(X_train , y_train,"Ridge")
op=rf.predict(X_test1)

yo1 = pd.DataFrame(columns=["Id","SalePrice"])
yo1["Id"]=raw_test.Id.values
yo1["SalePrice"]=op

yo1.to_csv('predictions_Ridge_cross.csv',index=False)
print ("k-fold␣validation␣Ridge" ,op);
```

```
#############################################Ridge cross validation ##########
############################################### Lasso ##########################
#0.94
reg = Lasso(alpha = 0.1)
reg.fit(X1,Y1);
op=reg.predict(X_test1);


yo1 = pd.DataFrame(columns=["Id","SalePrice"])
yo1["Id"]=raw_test.Id.values
yo1["SalePrice"]=op

yo1.to_csv('predictions_Lasso.csv',index=False)
print (op);
############################################### Lasso ##########################
############################################### Lasso Cross Validation ########
#0.57
rf=model_generation(X_train , y_train,"Lasso")
op=rf.predict(X_test1)

yo1 = pd.DataFrame(columns=["Id","SalePrice"])
yo1["Id"]=raw_test.Id.values
yo1["SalePrice"]=op

yo1.to_csv('predictions_Lasso_cross.csv',index=False)
print ("k-fold_validation␣Lasso" ,op);

############################################## Lasso Cross Validation #########
############################################## Neural Network ##################
#0.218
reg = MLPRegressor(hidden_layer_sizes=(100, ), activation='relu',learning_r
reg.fit(X1, Y1)
op=reg.predict(X_test1);


yo1 = pd.DataFrame(columns=["Id","SalePrice"])
yo1["Id"]=raw_test.Id.values
yo1["SalePrice"]=op

yo1.to_csv('predictions_nn.csv',index=False)
print (op);
############################################## Neural Network ##################
############################################## Neural Network Cross Validation
#0.233
rf=model_generation(X_train , y_train,"Neural")
op=rf.predict(X_test1)

yo1 = pd.DataFrame(columns=["Id","SalePrice"])
yo1["Id"]=raw_test.Id.values
```

```
yo1["SalePrice"]=op

yo1.to_csv('predictions_nn_cross.csv',index=False)
print ("k-fold_validation_Neural" ,op);



################################################## Neural Network Cross Validation
################################################# Kernelized ridge regression #####

reg = KernelRidge(alpha=1, kernel='linear', gamma=None, degree=3, coef0=1,
reg.fit(X1, Y1)
op=reg.predict(X_test1);


yo1 = pd.DataFrame(columns=["Id","SalePrice"])
yo1["Id"]=raw_test.Id.values
yo1["SalePrice"]=op

yo1.to_csv('predictions_Kernelized.csv',index=False)
print (op);
#################################################  Kernelized ridge regression ####
################################################# Cross Validation Kernelized ridge
rf=model_generation(X_train , y_train,"kernel")
op=rf.predict(X_test1)

yo1 = pd.DataFrame(columns=["Id","SalePrice"])
yo1["Id"]=raw_test.Id.values
yo1["SalePrice"]=op

yo1.to_csv('predictions_kernel_cross.csv',index=False)
print ("k-fold_validation_kernel" ,op);


################################################# Cross Validation Kernelized ridge

################################################# SVM ##################################

reg = svm.SVR() # rbf, linear
#print (Y1)
reg.fit(X1, Y1)
op=reg.predict(X_test1);


yo1 = pd.DataFrame(columns=["Id","SalePrice"])
yo1["Id"]=raw_test.Id.values
yo1["SalePrice"]=op

yo1.to_csv('predictions_svm.csv',index=False)
```

```
print (op);

############################################# SVM ##############################
############################################# SVM Cross #########################
rf=model_generation(X_train , y_train,"svm")
op=rf.predict(X_test1)

yo1 = pd.DataFrame(columns=["Id","SalePrice"])
yo1["Id"]=raw_test.Id.values
yo1["SalePrice"]=op

yo1.to_csv('predictions_svm_cross.csv',index=False)
print ("k-fold_validation␣svm" ,op);


############################################# SVM Cross #########################
############################################# Partial Least Square ##############

reg = PLSRegression()
reg.fit(X1, Y1)
op=reg.predict(X_test1);


yo1 = pd.DataFrame(columns=["Id","SalePrice"])
yo1["Id"]=raw_test.Id.values
yo1["SalePrice"]=op

yo1.to_csv('predictions7_pls.csv',index=False)
print (op);

############################################# Partial Least Square ##############
############################################# Partial Cross valid ###############
rf=model_generation(X_train , y_train,"partial")
op=rf.predict(X_test1)

yo1 = pd.DataFrame(columns=["Id","SalePrice"])
yo1["Id"]=raw_test.Id.values
yo1["SalePrice"]=op

yo1.to_csv('predictions_partial_cross.csv',index=False)
print ("k-fold_validation␣Partial" ,op);

############################################# Partial Cross Valid ###############
############################################# stocastic gradient decent ########
# for more than 10000 sample

reg = SGDRegressor(alpha=0.6,n_iter=1000);
reg.fit(X1, Y1)
op=reg.predict(X_test1);
```

```
yo1 = pd.DataFrame(columns=["Id","SalePrice"])
yo1["Id"]=raw_test.Id.values
yo1["SalePrice"]=op

yo1.to_csv('predictions_SGD.csv',index=False)
print('SGSSSSSSSSSSSSSSSSSSS')
print (op);
print (X1.shape)

############################################ stocastic gradient decent #########

############################################ Baysian regression ################
reg = linear_model.BayesianRidge()
reg.fit(X1, Y1)
op=reg.predict(X_test1);


yo1 = pd.DataFrame(columns=["Id","SalePrice"])
yo1["Id"]=raw_test.Id.values
yo1["SalePrice"]=op

yo1.to_csv('predictions_bayesian.csv',index=False)
print (op);

############################################ Baysian regression ################
############################################ Baysian Cross #####################
rf=model_generation(X_train , y_train,"Bay")
op=rf.predict(X_test1)

yo1 = pd.DataFrame(columns=["Id","SalePrice"])
yo1["Id"]=raw_test.Id.values
yo1["SalePrice"]=op

yo1.to_csv('predictions_Bay_cross.csv',index=False)
print ("k-fold_validation Baysian" ,op);

############################################ Baysian Cross #####################

##############################BOOSTING STARTS##################################
from sklearn.linear_model import Ridge, RidgeCV, ElasticNet, LassoCV, Lasso
from sklearn.model_selection import cross_val_score
from scipy.stats import skew
raw_train_boost = pd.read_csv('train.csv')
raw_test_boost = pd.read_csv('test.csv')
all_data = pd.concat((raw_train_boost.loc[:,'MSSubClass':'SaleCondition'],r

#log transform the target:
```

```
raw_train_boost["SalePrice"] = np.log1p(raw_train_boost["SalePrice"])

#log transform skewed numeric features:
numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index

skewed_feats = raw_train_boost[numeric_feats].apply(lambda x: skew(x.dropna
skewed_feats = skewed_feats[skewed_feats > 0.75]
skewed_feats = skewed_feats.index

all_data[skewed_feats] = np.log1p(all_data[skewed_feats])

all_data = pd.get_dummies(all_data)

#filling NA's with the mean of the column:
all_data = all_data.fillna(all_data.mean())

#creating matrices for sklearn:
X_train_b = all_data[:raw_train_boost.shape[0]]
X_test_b = all_data[raw_train_boost.shape[0]:]
y_b = raw_train_boost.SalePrice
def rmse_cv(model):
    rmse= np.sqrt(-cross_val_score(model, X_train_b, y_b, scoring="neg_mea
    return(rmse)
model_lasso = LassoCV(alphas = [1, 0.1, 0.001, 0.0005]).fit(X_train_b, y_b)

rmse_cv(model_lasso).mean()
import xgboost as xgb

dtrain = xgb.DMatrix(X_train_b, label = y_b)
dtest = xgb.DMatrix(X_test_b)

params = {"max_depth":2, "eta":0.1}
model = xgb.cv(params, dtrain,  num_boost_round=500, early_stopping_rounds=

model.loc[30:,["test-rmse-mean", "train-rmse-mean"]].plot()

model_xgb = xgb.XGBRegressor(n_estimators=360, max_depth=2, learning_rate=
model_xgb.fit(X_train, y)

xgb_preds = np.expm1(model_xgb.predict(X_test))
lasso_preds = np.expm1(model_lasso.predict(X_test))

predictions = pd.DataFrame({"xgb":xgb_preds, "lasso":lasso_preds})
predictions.plot(x = "xgb", y = "lasso", kind = "scatter")

preds = 0.7*lasso_preds + 0.3*xgb_preds

solution = pd.DataFrame({"id":test.Id, "SalePrice":preds})
solution.to_csv("ridge_sol.csv", index = False)
```

###########################BOOSTING ENDS###################################

3.3 Comparison of result using different machine learning techniques

These are the machine learning techniques which we have used for predicting the sales price of house:

1. Ridge Regression
2. Kernelized Ridge Regression
3. Lasso Regression
4. Feed Forward Neural Network
5. Support Vector Machines
6. Partial Least Square
7. Stochastic gradient decent
8. Bayesian regression
9. Boosting
10. Applied cross validation to all the above machine learning models.

3.3.1 Results

Without feature Engineering

Model	Error (Without Cross Validation)	Error (With Cross Validation)
Ridge Regression	0.93136	0.55967
Kernelized Ridge Regression	0.83350	0.43901
Lasso Regression	0.94388	0.57851
Feed Forward Neural Network	0.21314	0.21956
Support Vector Machines	0.41890	0.41881
Partial Least Square	1.79965	NA
Stochastic gradient decent	NA	NA
Bayesian regression	0.15672	0.17316
Elastic Net	0.15298	0.55967
Boosting	0.12087	NA

With feature Engineering

Model	Error (Without Cross Validation)	Error (With Cross Validation)
Ridge Regression	0.21572	0.20987
Kernelized Ridge Regression	0.20620	0.43901
Lasso Regression	0.21572	0.20987
Feed Forward Neural Network	0.30532	0.33858
Support Vector Machines	0.41890	0.41881
Partial Least Square	0.35663	0.35726
Stochastic gradient decent	NA	NA
Bayesian regression	0.21633	0.30827
Elastic Net	0.21636	0.20859

3.3.2 Inference

1. Boosting is giving best result without applying feature selection as it uses random forest that already implements the feature selection techniques.

2. The Best Result we are getting is 0.12087. Our results are among top 25% on kaggle out of 2280 participants.

3. Both Ridge and Lasso results gets better when we applied Cross Validation Technique on them as shown in above table. Here We are using K-fold Cross Validation. Using this method we are selecting model which is giving min root mean squared error among k iteration.

4. Both Ridge and Lasso results gets better as we applied feature engineering on the data set. Error reduced by 76.83% in case of ridge and 74.11% in case of Lasso . This is because unlike other models(i.e. Neural Network, Boosting etc) these models does not internally do feature selection. So the other unimportant feature effect the predicted values.

5. There is no effect of Feature Engineering on Neural network. As it does so internally. But the result is better in case when no feature engineering is applied.

6. SVM model 's result is same in all cases. Also its result are better than Ridge and Lasso when there is no feature engineering is applied.

7. Bayesian give best result when no feature engineering and cross validation is applied.

8. Stochastic gradient decent we could not apply since it requires more than 10,000 training data set examples.

Chapter 4

Contribution

All have done work equally.

Bibliography

Dubin, Robin A. Predicting house prices using multiple listings data. 1998.

P. B. Jae Kwon Bae. Using machine learning algorithms for housing price prediction: The case of Fairfax County, Virginia housing data. 2015.

Limsombunchao, V. House price prediction: hedonic price model vs. artificial neural network. 2004.

X. J. Y. Yubiao Wang. Real estate price forecasting based on SVM optimized by PSO. 2013.