CALIFORNIA STATE UNIVERSITY SAN MARCOS

PROJECT SIGNATURE PAGE

PROJECT SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

PROJECT TITLE: ZILLOW HOME VALUE PREDICTION USING XGBOOST

AUTHOR: CHANNAMALLIKARJUN SIDDARAMAPPA ROLLI

DATE OF SUCCESSFUL DEFENSE: 10/23/2019

THE PROJECT HAS BEEN ACCEPTED BY THE PROJECT COMMITTEE IN
PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF
SCIENCE IN COMPUTER SCIENCE.

Xin Ye	Xin Ye	2/5/2020
PROJECT COMMITTEE CHAIR	SIGNATURE	DATE
Xiaoyu Zhang		2/10/2020
PROJECT COMMITTEE MEMBER	SIGNATURE	DATE
PROJECT COMMITTEE MEMBER	SIGNATURE	DATE

Zillow Home Value Prediction (Zestimate)
By Using XGBoost

BY,

Channamallikarjun Rolli

rolli014@cougars.csusm.edu

California State University San Marcos

Abstract

A home is one of the largest/expensive purchase, people make in their lifetime. Making sure homeowners monitor their property in a trusted way is incredibly important. The Zestimate model was developed to give detailed information about home and market value of the home to the first-time consumers at free of cost. In this paper, I analyzed the real estate property prices in three counties in California (Los Angeles, Ventura, Orange). The information on the property listing was taken from Kaggle.com. I predicted sold price and asking prices of home properties based on features such as bedroom count, bathroom count, geographical location etc. I used gradient boosting models, XGBoost. **Mean Absolute Error** is how the results are evaluated between the log error (actual log error and predicted log error). The shape of the training data set is 90275,3 and that of the properties data set is 2985217,58. The shape of the merged dataset is 90275, 60. Target variable for this project is variable "log error". The log error has a normal distribution of data. The detailed prediction questions, the analysis of the real estate property, the testing and validation for other different algorithms have been presented in this paper. In addition, I will discuss my approach and methodology.

Keywords: XGBoost, Zestimate, Mean Absolute Error, House Value

Acknowledgement

Thank you to Dr. Xin Ye for guiding me throughout the entire researching process, Dr. Xiaoyu Zhang for his valuable comments and to my friends and family for supporting me during my Master's time in California State University, San Marcos for offering me the support and opportunity to work on this project.

Contents

List of Figures

List of Figures

Introduction and Background

Real estate property prices are linked with our economy. Although we have a vast amount of properties listing, we do not have an accurate prediction model to predict or measure the real estate price. In three counties in California (like Ventura, Orange and Los Angeles) there are around 2985217 current listings at Kaggle.com [1]. The dataset provided by Kaggle.com has hundreds of features, ranging from room count, bathroom count to living area etc. With this dataset it should be enough to establish a model such as regression to predict the real estate price accurately.

A Real estate property's estimated value is very important in transactions such as loans, marketability and sales. Traditionally, real estate property estimator used to determine the price of the real estate. But the disadvantage with this method is that the estimator is more likely to be biased and can change the price of property on vested interest of seller/buyer or from the lender[2]. Therefore, to get rid of such method, an automated prediction model can be used as independent source which can be less biased.

For the first time property buyer's prediction model's suggestion will be quite useful. With the automated prediction model, they can find underpriced or overpriced real estate property values on the market.

The Boston real estate dataset is commonly used to analyze the real estate price using regression model. The dataset is strongly dependent on the size and socio geo location. The basic algorithm such as linear regression achieves 0.113 accuracy for intrinsic feature and socio geographical features [3].

In this learning paper, the prediction was made for selling prices of the real estate using gradient boosting models such as XGBoost Regression method. Predicted the mean absolute error with an error of 0.05303 using an XGB Regression method. Below I have discussed the detailed prediction questions, the analysis of the real estate property, the testing and validation for other different algorithms have been presented in this paper. In addition, I will discuss my approach and methodology.

Related Work

Real estate is almost the most expensive thing people may purchase in his or her lifetime, so evaluating the house price could always affect people's decisions in their life. In order to ensure that real estate property owners have a trusted and better way to monitor this property, Zillow offers Zestimate to help consumers make decisions with multiplied information about homes and housing market. Recent project uses valid home values data provided by Zillow to develop predictions model to predict about future sale price of real estate property and push the accuracy of Zestimate even further. For decades, researchers have been looking for ways to estimate and predict housing prices. In this section, I have discussed the previous prediction models and research's that happened for the housing price in the market. To give an overview of the previous prediction model, I have focused on models which are different from my model. In a most recent time, Bhuiyan et al. [4], presented a model to show how fast the real estate property will be sold once its on the market listings. They extracted the data from Trulia and got the dataset from five cities in Indianapolis. Like our dataset, their data contained room count, bath count, fire place count etc. Using these features, they predicted the probability of survival of the real estate property. Which was the key role for their model.

2.1. Automatic house price prediction

In another work in real estate property field by Lim et al. [5], predicted condominium prices. These properties are special kind of real estate housing which have built-in features like clubhouse, swimming pool barbecue area and gym. Researchers in this project these properties not only on features like room count, garage count, bathroom count etc., but also on features such as CPI (consumer price index), real interest rate, GDP (gross domestic product), PLR (prime lending rate), CAP (condominium asking prices). Lim predicted CPI and CAP of condominium using different models such artificial neural network (ANN), multiple regression analysis (MRA) and autoregressive integrated moving average (ARIMA). They

concluded that for predicting CPI, ANN model is better ARIMA and to predict CAP, MRA is better than other models.

2.2 Machine learning Algorithms

2.1.1 Linear Regression

Linear regression is the first model [6][7]. Linear regression predicts function P as a linear function of input x. It finds an intercept w_0 and coefficient variable w = (w_1,…..,w_m) such that

$$P(x) = w0 + w^T x = w_0 + w_1 x_1 + ... + w_m x_m$$

It's accomplished through using least squares approach. Here the price is minimized using sum of squared differences is between predicted price and actual price.

Linear regression was chosen as first model due to its wide spread use and simplicity in this field.

2.1.2 Gradient Boosting

The third model which I used was Gradient boosting [8]. Ensemble technique is used by this algorithm. Multiple decision trees are built and summed to get the prediction F(x). We can represent it as [9]

$$F(x) = tree1(x) + tree2(x) + \ldots$$

The discrepancy between the value is covered by learning from the previous tree. For example, if we have two trees such that

$$F(x) = tree1(x) + tree2(x)$$

Then the subsequent tree will be trained it minimizes the error from the previous tress [10]. Ideally, it should look like below

$$F(x) + tree3 = P(x)$$

At last the final prediction, the model removes the discrepancy target value and current value. This is called residual.

$$R(x) = P(x) - F(x)$$

The iterative process is used in developing the tree.

2.1.3 Random forest Regression:

I used Random Forest as my next model due to its flexibility and very easy to use. Random forest predicts great result even without using hyper-parameter almost all the time. This algorithm is widely used because of its simplicity and it can be used for regression problems and classification problem.

Explanation of The RandomForest algorithm is as follows [11]:

$$MSE = \frac{\sum_{Samples}(Price_{sample}{}^2 - value^2}{tot\ number\ of\ samples}$$

$$Weighted\ MSE = \frac{NumSamplesNode1 * MSE1 + NumSamplesNode2 * MSE2}{NumSampleNode1 + NumSampleNode2}$$

1) Features are evaluated at every node for the better split. Mean square error is achieved through this split between predicted value and actual value of the property.
2) Levels of tree(depth) can be specified using scikit-learn as max_depth in Random Forest.
3) Minimum samples can be specified using min_samples_leaf. With the help of this parameter we can limit overfitting. We can also used max_features to limit overfitting.
4) To specify the tree numbers, we can use n_estimators. Using 10-40 tress is always optimal for the better result.
5) To specify the parallel task to run in the CPU, we can specify n_jobs. -1 specifies that model uses all the CPU.

Problem Statement and Data Collection

The problem is to build a model which predicts the housing price using different features such bedroom count, bathcount, kitchensize, squarefeet. The dataset has been extracted from Kaggle.com and it has a property listing from three counties in California (Ventura, Los Angeles and Ventura). The target which has to be predicted is little difference depending on the prediction usage.

For example, the underprice or overprice can be determined currently by extracting the asking property of the asking price. We can determine underprice if the asking price is less than the prediction value. Similarly, we can determine overprice if the asking price greater that the prediction model [12].

All the real estate transactions in the U.S. are publicly available. Kaggle.com has provided the data of properties from three counties in California (Ventura, Los Angeles and orange) of 2016. They provided the train data of all the transaction.

The shape of the training data set is 90275,3 and that of the properties data set is 2985217,58. The shape of the merged dataset is 90275, 60. Target variable for this competition is variable "log error". There are few outliers. The log error is normally distributed. Out of the 60 variables; 53 are float, 1 date time, 5 objects and 2 ints. And, also, we will have to convert objects to integers by transforming the data before feeding the same to the model.

Data Analysis

4.1 Data Exploration:

We can observe from the dataset that 90% of the featuers in the data set are float variables. The breakup is as follows: int64→1, float64→53, datatime64[ns]→1 and object→5. When performing missing data analysis, we can observe that most of the variables have a high ratio of missing data. There are 25 variables which have a missing ratio of > 70%. From correlation analysis [Figure 2], we can observe that the following pairs are highly correlated (bathroomcnt, calculatedbathnbr), (calculatedfinishedsquarefeet and

finishedsquarefeet12), (rawcensustractandblock, censustrackandblock) and (taxvaluedollarcnt, taxamount). The data has been divided into continuous, discrete and categorical data. By plotting histograms on the continuous variables, by observation we can find that most of the data is distributed normally. By plotting countplots on the discrete variables we can see that the data some of the values have high counts for discrete variables. By plotting barplots on categorical variables we can observe that some of the columns have NaN values and we remove the columns with greater than 70% NaN(Figure 3). The dataset is available in the Kaggle website under the competition “Zillow Prize: Zillow’s Home Value Prediction (Zestimate)”. The shape of the training data set is 90275,30 and that of the properties data set is 2985217,58. The shape of the merged dataset is 90275, 60. Target variable for this project is variable "logerror".

Sample Data:

	parcelid	logerror	transactiondate	airconditioningtypeid	architecturalstyletypeid	basementsqft	bathroomcnt	bedroomcnt	buildingclasstypeid
0	11016594	0.0276	2016-01-01	1.0	NaN	NaN	2.0	3.0	NaN
1	14366692	-0.1684	2016-01-01	NaN	NaN	NaN	3.5	4.0	NaN
2	12098116	-0.0040	2016-01-01	1.0	NaN	NaN	3.0	2.0	NaN
3	12643413	0.0218	2016-01-02	1.0	NaN	NaN	2.0	2.0	NaN
4	14432541	-0.0050	2016-01-02	NaN	NaN	NaN	2.5	4.0	NaN

5 rows × 60 columns

Figure 1: Sample data

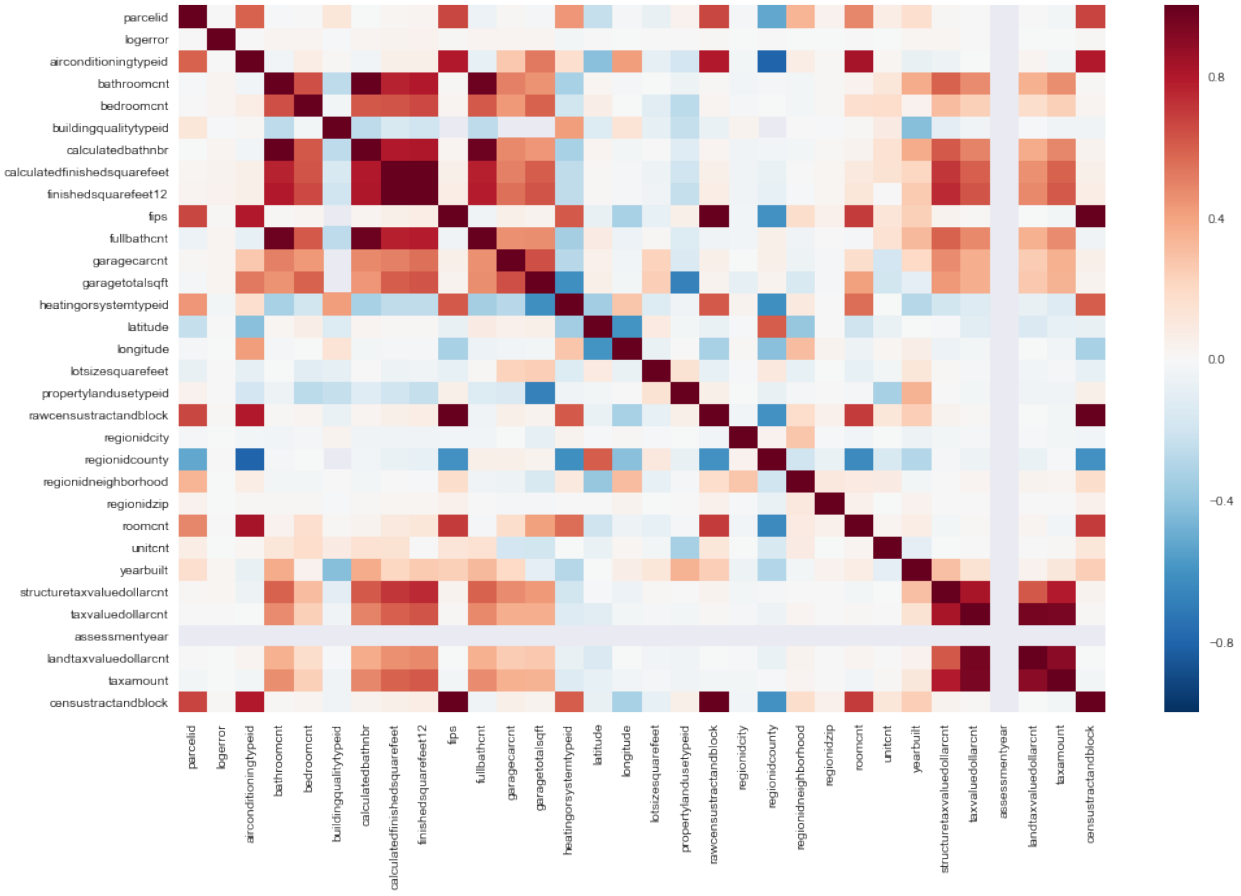

Figure 2: Correlation Analysis

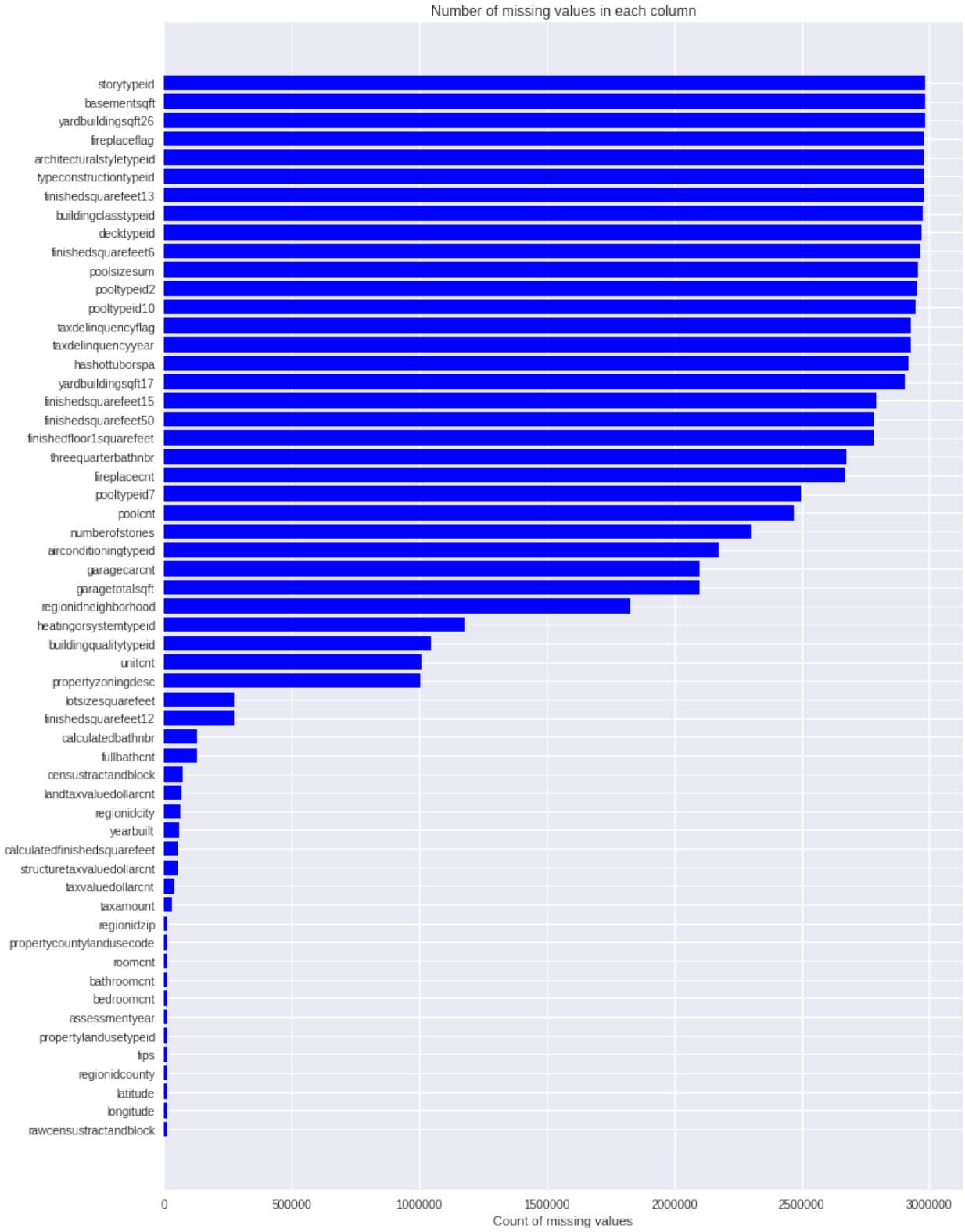

Figure 3: Number of missing values in each column for every feature

The target variable ‘logerror’ is normally distributed.

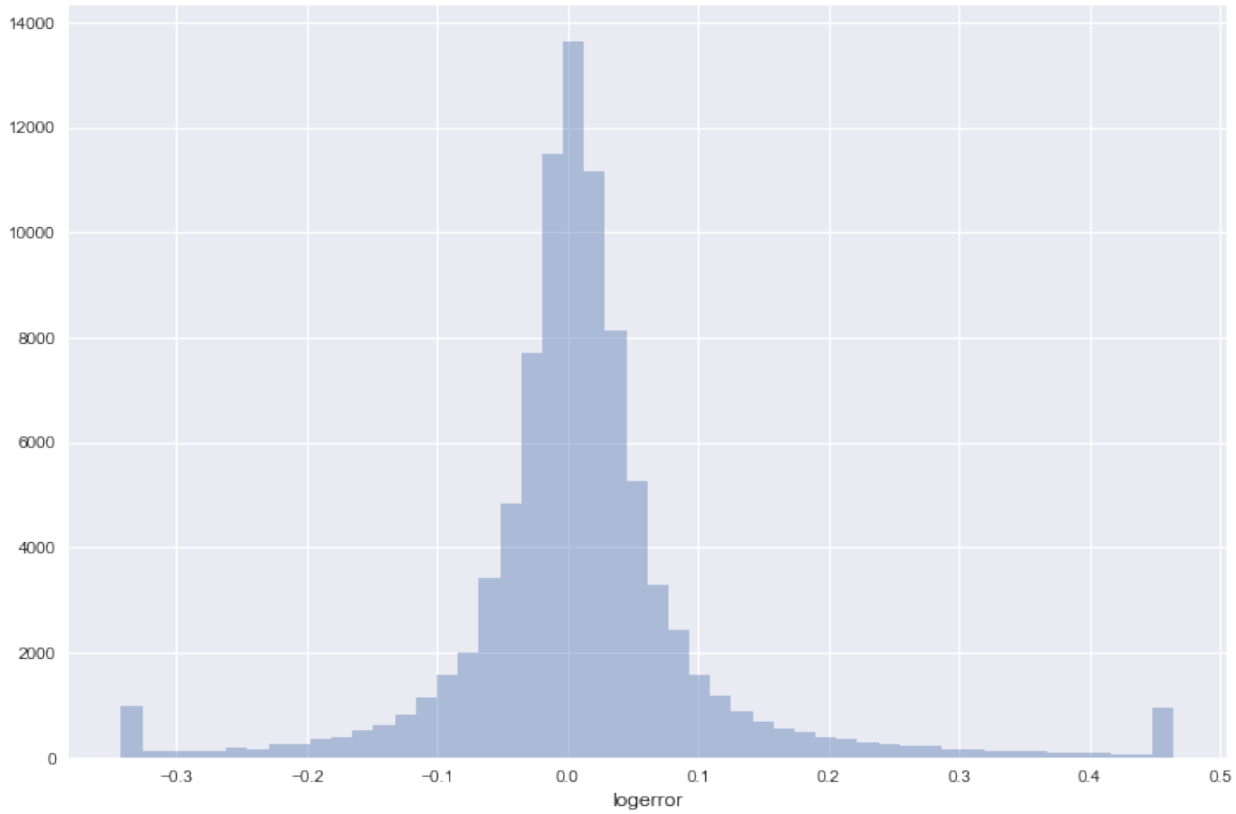

Figure 4: Shows the logerror: Mostly Normally Distributed

4.2 Exploratory Visualization

The data has been divided into continuous (Figure 5, 6, 7, 8, 9), discrete (Figure 10,11,12,13) and categorical data. Histograms have been plotted for count plots for discrete variables, barplots for categorical variables and heatmap for correlation analysis have been plotted. Most of the variable have been visualized and thoroughly analyzed. Exploring few of the variables from the dataset:

Histogram of continuous variable: taxamout (Figure 5). We can observe that the mean tax amount is around 5983.97, with a range of [49.08, 321936.09].

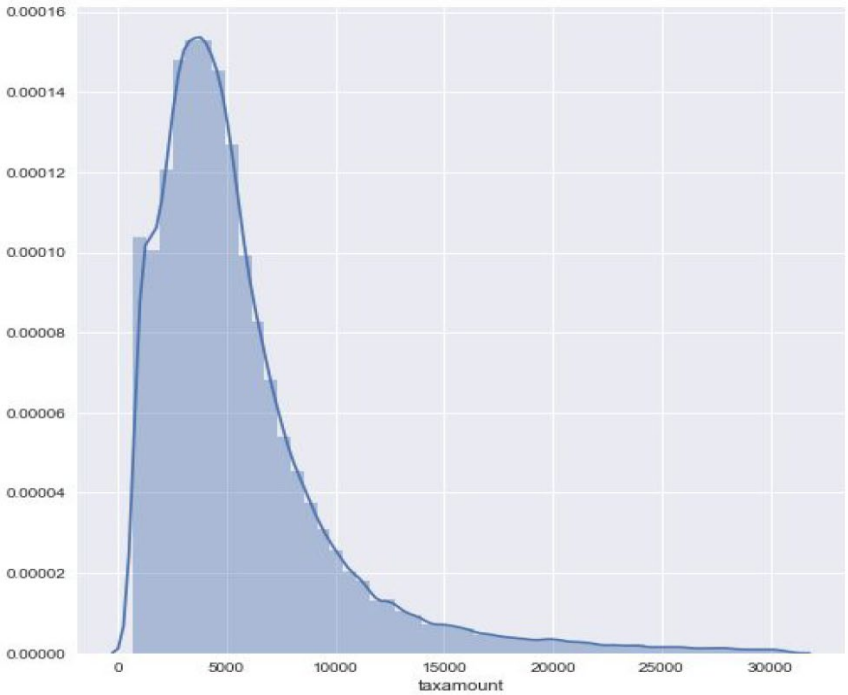

Figure 5: displot of continuous feature: taxamout

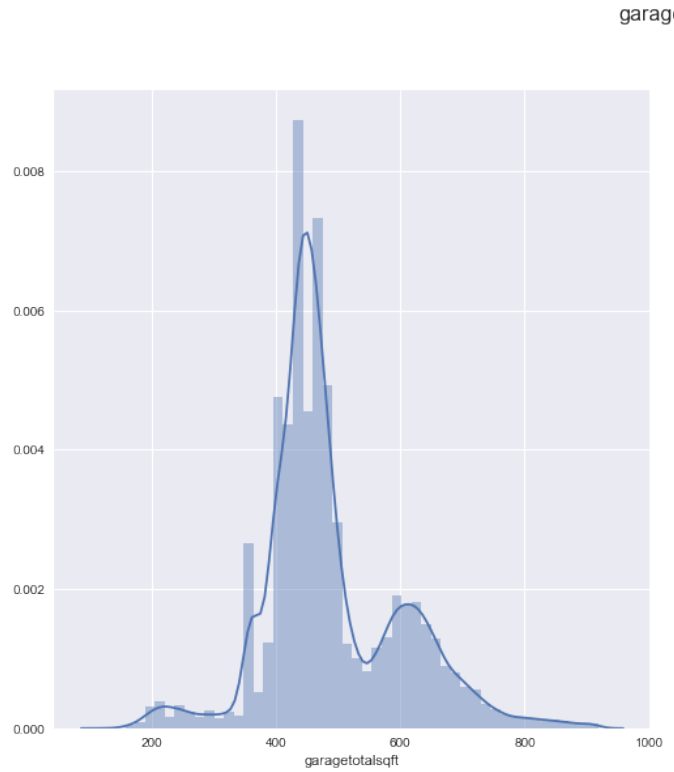

Figure 6: displot of continuous feature: garagetotalsqft

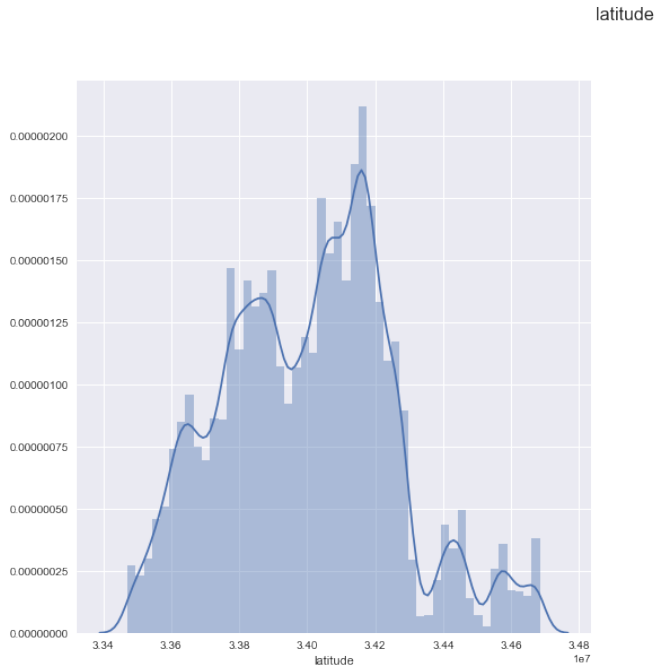

Figure 7: displot of continuous feature: latitude

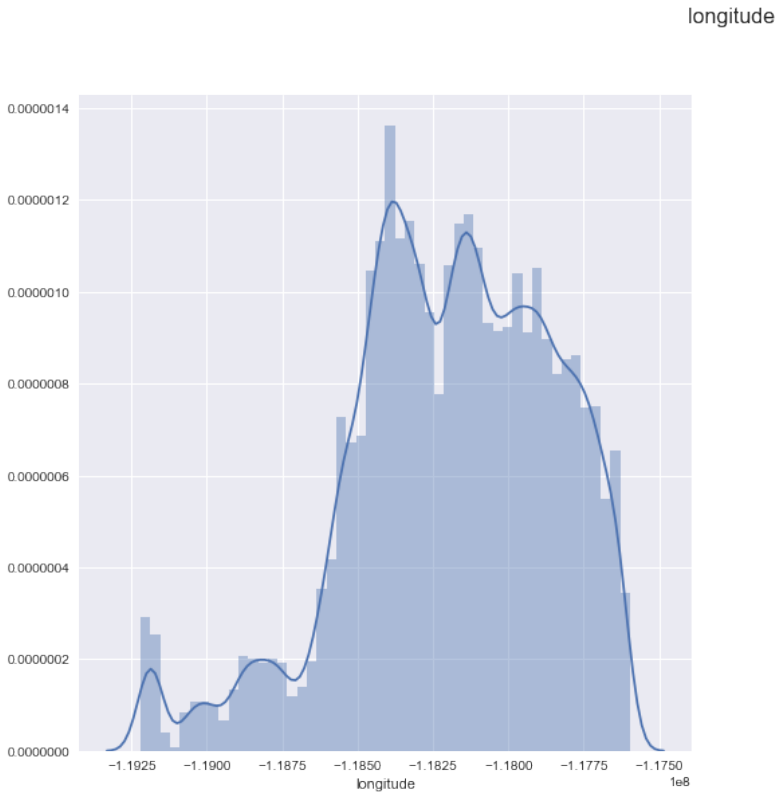

Figure 8: displot of continuous variable: longitude

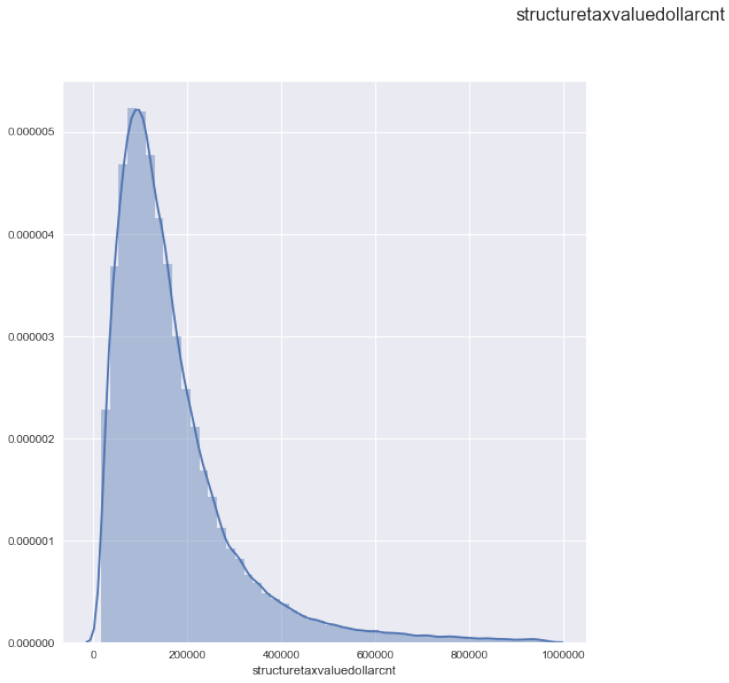

Figure 9: displot of continuous feature: structuretaxvaluedollarcnt

Count plot of discrete variable: bathroomcnt. This indicates that most of the house have 2 or 2.5 or 3 bathrooms.

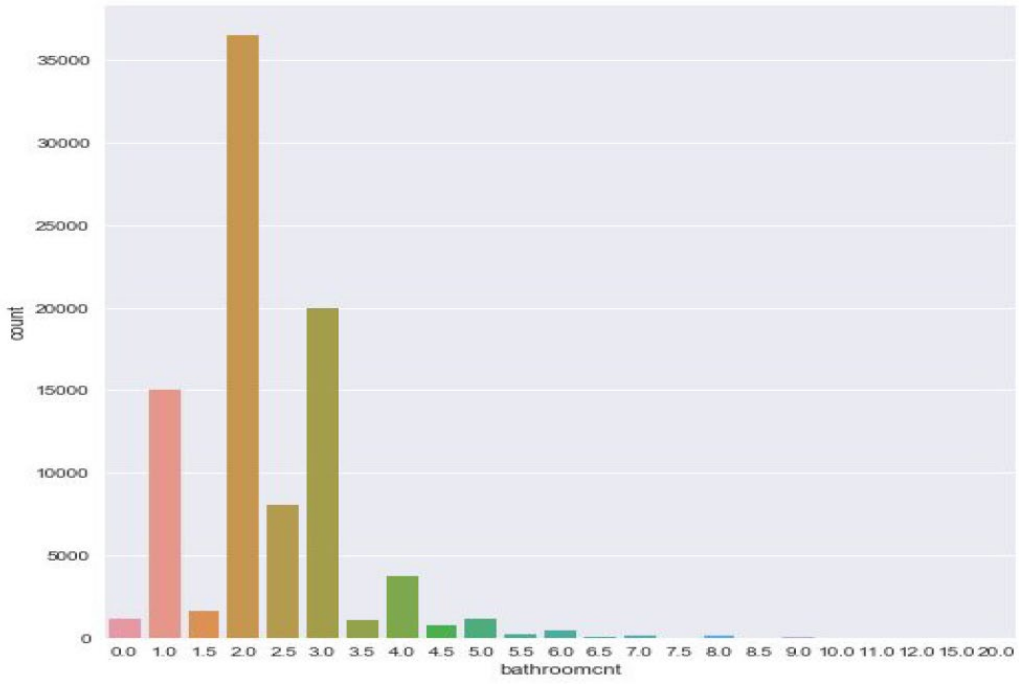

Figure 10: Count plot of discrete variable: bathroomcnt

Bar plot of discrete variable: propertylandusetypeid. We can clearly observe that most of the data (92%) has propertylandusetypeid as either 261 or 266.

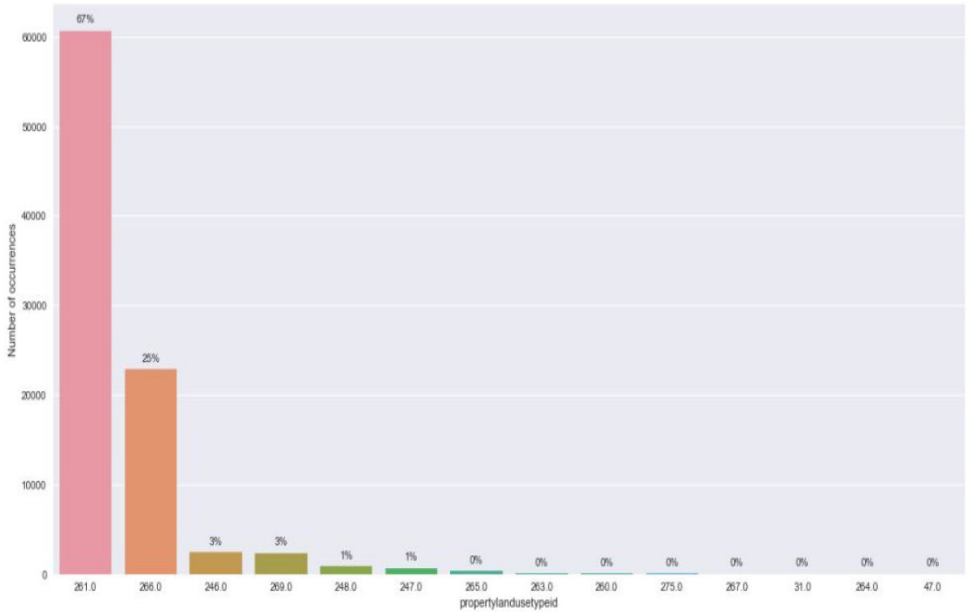

Figure 11: Bar Plot of discrete variable: propertylandusetypeid

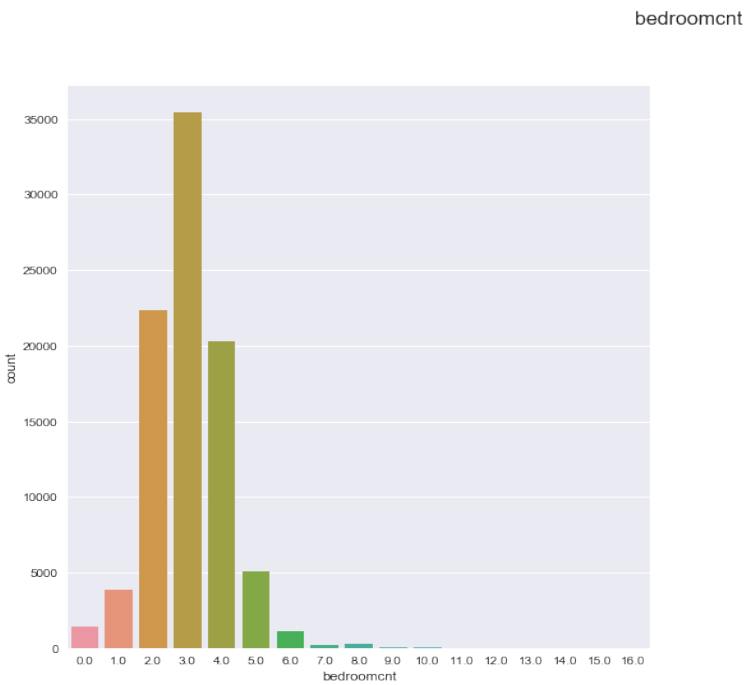

Figure 12: Bar plot of discrete variable: bedroomcnt

Discrete variable: garagecarcnt. This discrete variable is another important value and from the graph clearly indicate that most of the houses have 2 garages.

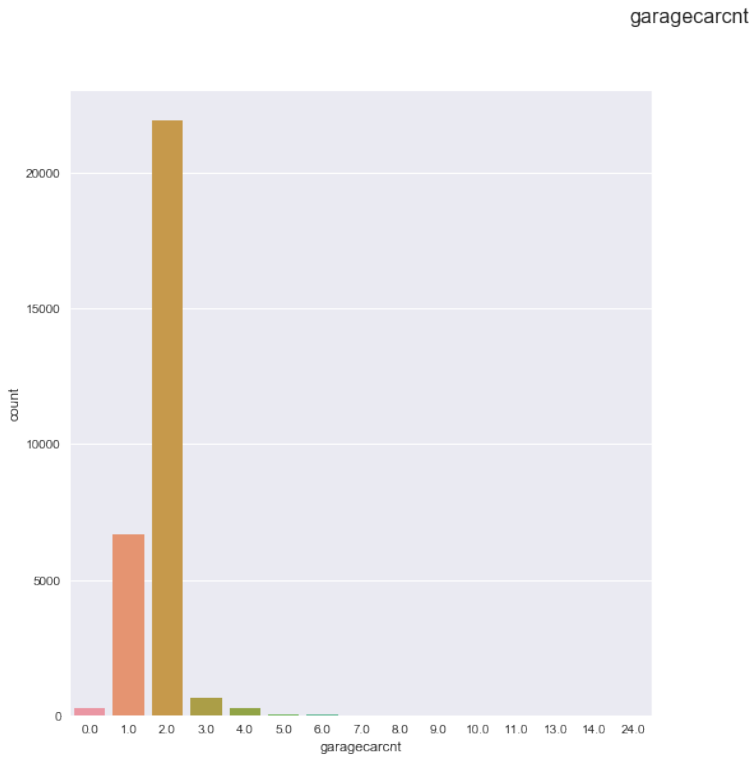

Figure 13: Bar plot of discrete variable: garagecnt

XGBoost algorithm

XGBoost algorithm was used for solving the problem. XGBoost[13] is an implementation of gradient boosted decision tree and it stands as eXtreme Gradient Boosting [Figure 8]. XgBoost is known for its flexibility, performance and speed. Compared to another model XGBoost is fast. This model is best used for tabular dataset and structured and it works best for classification and regression models. XGBoost is enabled with parallel processing making it at least ten times faster than any other tree-based models. It avoids overfitting by Regularization. XGBoost handles Missing values internally. Due, to all above mentioned features XGBoost algorithm was used - fast, handles regression, optimized, good on tabular datasets, flexible parameters, handles missing

values etc. The ensemble tree model (Figure 14) is a group of regression and classification tress. The following formula and figure [Figure 15] are derived from XGBoost documentation which show how a person based on inputs such as age, gender, occupation etc. can be classified as a person who likes computer games or not.

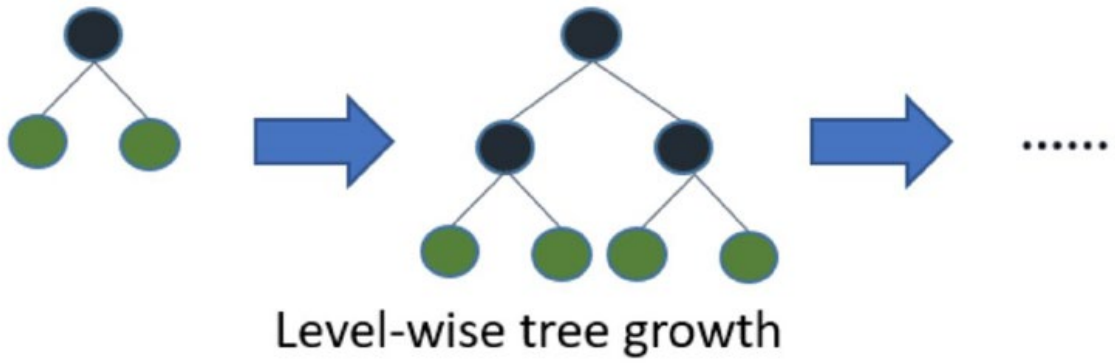

Figure 14: XGBoost Level-wise tree growth[13]

Since XGboost is a library we can download and install and get different kind of interfaces [13]. The XGBoost library implements the gradient boosting decision tree algorithm. The algorithm has many names like multiple additive regressor tree, stochastic model tree etc. XGBoost model is completely focused on the calculation speed and performance of the model. XGBoost gives us wide range for computing:

- Parallel constructing the tree using CPU core while training.
- We can do distributed computing for large dataset using group of machines.
- The dataset that won't fit in the memory, we can use Out of core computing.
- Optimization of Cache to make use of hardware.

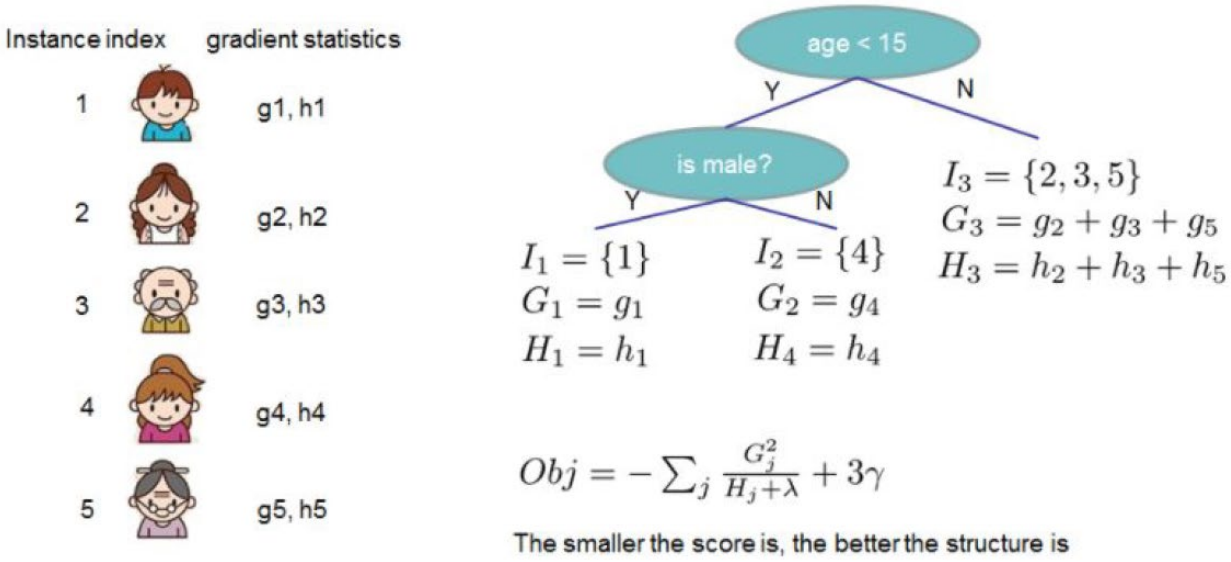

Figure 15: XGBoost Tree example[14]

Let's explain algorithms a little bit further. Boosting is to a group of algorithms that can change the weak learns into strong learners. 1) base learner assigns equal weight to each observation. 2) It will assign higher weights to observations having prediction errors, if there are any errors caused by first base learning algorithm. 3) repeat second step until higher accuracy is achieved or limit is reached. 4) combine all the weak learners and generate a strong learner which improves prediction power. Let's talk about Gradient Tree Boosting first, then I will get into XGBoost and lightGBM. Gradient boosting has 3 elements: 1) A loss function 2) A weaker learner 3) An additive model to add weaker learners. Regress tree is used so that subsequent model outputs can be added together and fix the residual error in the prediction. Trees are added one at a time. Gradient boosting can overfit very quickly since it's a greedy algorithm. This is where XGBoost comes into the picture. XGBoost is built on base Gradient Boosting Model with the goal of improving speed and accuracy of original model. Let's talk about what XGBoost does different from Gradient Boosting Trees. First, XGBoost uses second-order gradient of the loss function (add to the first-order gradient) based on Taylor expansion of the loss function. Taylor expansion of different types of loss functions can be plugged into the same algorithm for greater generalization. Second, XGBoost transforms the loss function

into a complicated objective function which has regularization terms. This new transformation - extension of the loss function adds penalties when it adds new decision tree leaves to the model. Penalties are proportional to the size of the leaf weights. This prevents the growth of the model and prevents overfitting. In addition to accuracy improvement, it also improves speed a lot because XGBoost uses cache and memory optimization. XGBoost performs so much better than Gradient Boosting Trees [14].

Evaluation Metrics

Mean Absolute Error is the way the results are evaluate between the predicted value and the actual error [15]

$$MAE(y, \hat{y}) = \frac{1}{nsamples} \sum_{i=0}^{nsamples-1} |yi - \hat{y}i|$$

Where

Actual = yi

Predicted = $\hat{y}i$

Equation 1: Equation to calculate MAE

As, the problem is a regression problem. The evaluation of the solution is based on negative MEA, '**neg_mean_absolute_error**' parameter is chosen for model evaluation. This 'regression' scoring metric provided by scikit is used to measure the negative MAE value[15]. The metric measures the distance between the model and the data and in case of 'neg_mean_absolute_error'**.**

The log error is defined as

logerror=log(Zestimate)−log(SalePrice)

Difference between two continuous variables is negative mean absolute error. **The solution for the problem will be written in python using scikit and other packages**. 'neg_mean_absolute_error' is a good measurement for this project because of the nature of the datasets. The targeted value - logerror distributes from -4 to 4. It ranges from negative

value to positive value. **Mean absolute error can take care of negative value issue.** It gives proper neg_mean_absolute_error' regardless of negative or positive data [16]

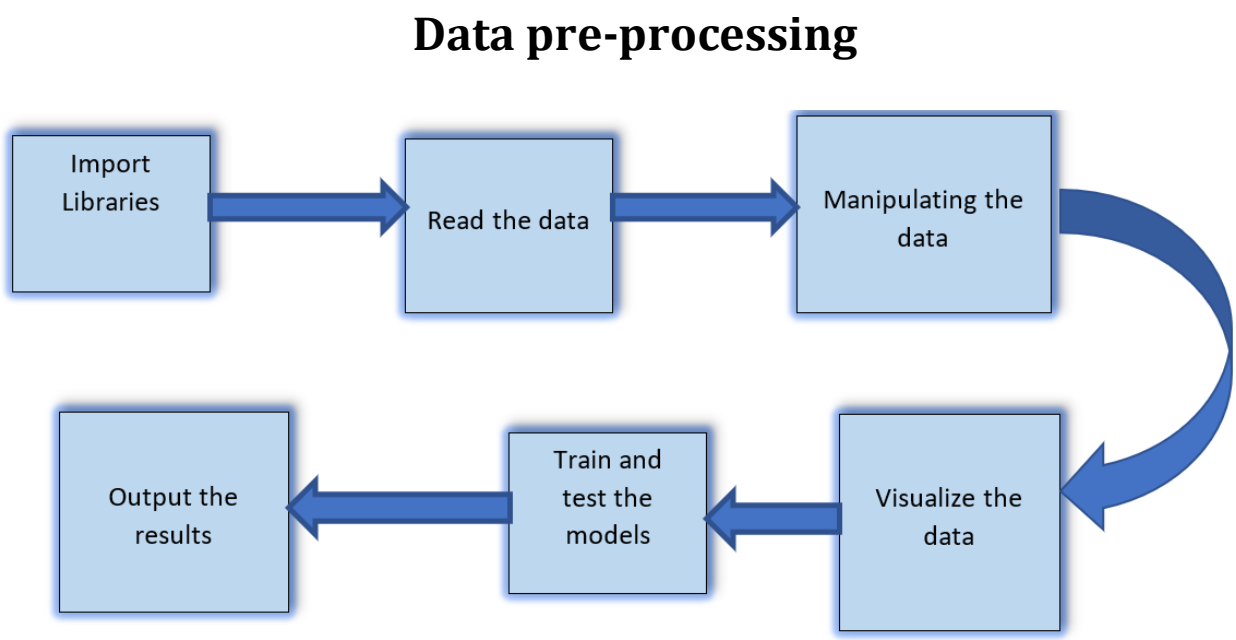

Figure 16: Flow Diagram

I used Jupyter notebooks and python scripts to get the scores for the models. Several libraries including sklearn, numpy, pandas, xgboost, and seaborn were used to (1) read the data, (2) manipulate the data, (3) visualize the data, (4) train and test the models, and (5) output the results.

7.1 Reading the data

Used panda library to load the dataset files. Zillow has provided us with properties_2016.csv and train_2016.csv. Read_csv panda function is used to read the data from csv files. To train a model, I first read the data from the "properties" dataset. Then, added new features to this dataset. Next, I read the "training" data and merged it with our augmented "properties" dataset.

7.2 Importing the libraries

a. **seaborn**
b. **Scipy**
c. **scikit-learn**
d. **pandas 0.20.3**
e. **Numpy**
f. **Matplotlib**

7.3 Missingness and Imputation

Before fitting a model to the data, we first need to perform a preliminary assessment of the dataset for missing values. The Zillow dataset contains features with lots of missing values. For example, some "number" features contained >70% NaN values.

The strategy for handling missing data was one of the key challenges of this project. The approach to this problem was to systematically assess each variable, in order to better understand the reason for missingness. These variables can be separated into two broad categories: first, those in which the reason for the NaN is implied by the data, and second, those for which the reason for the NaN is unknown.

For several of the variables with high missingness, I believe that missing data points simply implied that the property does not contain the item in question. These variables contain a distribution of ones, signifying that the property contains the item, intermixed with NaNs. These categories include topics such as tax delinquency, lot size, fireplace, pool, garage, basement, and hot tub; it makes intuitive sense that many of the properties in southern California do not have some or all these features. Thus, I imputed NaNs in these categories with zero to signify that the feature is not present for those observations.

I used mean imputation for the calculated finished square feet variable, as this variable is nearly complete, so imputing with the mean value has a negligible effect on the overall

distribution of the variable. I dropped the other floor space variables, as they are either entirely redundant or highly missing. For the location variables, missingness was also very low, so imputed by randomly sampling from the distribution of each variable.

Given the importance of the tax assessment features, I took care to make informed imputations for missing values. For properties with property taxes paid values, I divided these values by the median tax rate (these were fairly consistent) across all properties to get an estimated assessment value. Over 99.9% of these cases had zero bathrooms and zero bedrooms, so imputed the estimated assessment value as the land assessment and imputed zero for the building assessment. For properties with no taxes paid values, imputed the average building and land assessment values, grouped by zip code, number of bedrooms, and number of bathrooms - assuming that properties with the same number of beds/baths in the same zip code were likely to have similar land and building assessment values.

Data preprocessing also involved the following activities:

- ***I.*** Merging data file and properties file
- ***II.*** Cleaning data by dropping columns which have a missing ratio of more than 70%
  - ***a.*** 25 variables were dropped as part of this process.
- ***III.*** On correlation analysis, it was observed that the following pairs are highly correlated
  - ***a.*** bathroomcnt, calculatedbathnbr.
  - ***b.*** calculatedfinishedsquarefeet and finishedsquarefeet12
  - ***c.*** rawcensustractandblock, censustrackandblock
  - ***d.*** taxvaluedollarcnt, taxamount.
- ***IV.*** Variables: ‘propertyzoningdesc’ and ‘propertycountylandusecode’ which had random values were dropped before feeding the data to the models.
- ***V.*** As part of feature engineering new variables were added
  - ***a.*** New_Transaction_Month: Month of the transaction
  - ***b.*** New_LivingAreaProp: Proportion of living area derived by (calculatedfinishedsquarefeet) / (lotsizesquarefeet)

c. New_zip_count: Total number of housing in this area.

d. New_city_count: Total Number of housing in the city.

The below table represent all the selected features.

	Feature	Data Type
0	aircondtioningtypeid	float32
1	bathroomcnt	float32
2	bedroomcnt	float32
3	buildingqualitytypeid	float32
4	calculatedfinsishedsqaurefeet	Float32
5	fips	Float32
6	garagecarcnt	Float32
7	garagetotalsqft	Float32
8	heatingorsystemtypeid	float32
9	latitude	float32
10	longitude	float32
11	lotsizesquarefeet	float32
12	propertylandusetypeid	float32
13	regionidcity	float32
14	regionidcounty	float32
15	regionneighborhood	float32
16	regionidzip	float32
17	roomcnt	float32
18	unitcnt	float32
19	yearbuilt	float32
20	structuretaxvaluedollorcnt	float32
21	assessmentyear	float32
22	taxamount	float32
23	New_trasaction_month	Int64

	Feature	Data Type
24	New_LivingAreaProp	float32
25	New_zip_count	Int64
26	New_city_count	Int64

Table 1: Selected Features

Model Selection

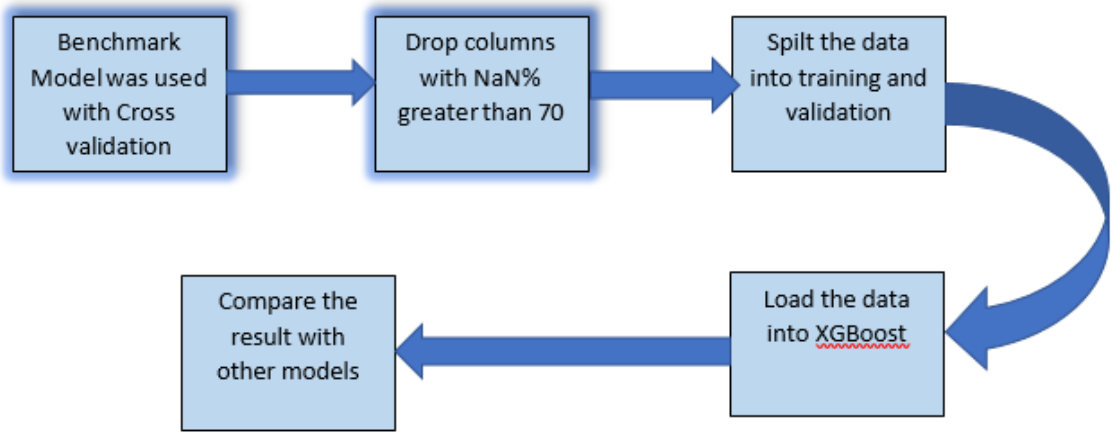

Figure 17: Flow diagram of Model Selection

After preprocessing the data, benchmark model [16], was used with cross validation. First, I dropped couple unimportant features from merged data set before I assigned it to training data and targeted values. Then I split up the training dataset at 90000 to make part of it as training set (60000) and another part of it as validation (30000) I loaded both final training dataset and targeted values into existing XGBoost program(xgboost 0.6 library) to speed up the performance.

Before training, couple parameters were picked up and passed to the model. Those parameters are min_data, objective, learning_rate, min_hessian, sub_feature, boosting_type, num_leaves, and metric. Min_data is the minimum number of data in one

leaf. This can be potentially to use to control overfitting. Benchmark model used 500 as min_data. This is a petty optimal choice. I tried lower and higher number of min_data, they all gave worse performance than 500. Regression was chosen for objective function because this is a regression supervised learning problem. Learning rate was set to 0.002. This can be used to control training speed. Lower rate lowers overfitting speed. Min_hessian means min number of hessians in a leaf for a split. It was set to 1. Higher value potentially decrease overfitting. Sub_feature is alias for feature_fraction. When this value is less than 1.0, it will make XGBoost randomly select part of features on each iteration. Sub_feature was set to 0.5. So, 50% of features were randomly selected before training each tree. This can be used to deal with training speed and overfitting. For boosting_type, gbdt was chosen. Gbdt means traditional gradient boosting decision tree. Nu_leaves are a total leaves in a tree. Num_leaves were set to 60. The last parameter is metric. Since I use neg_mean absolute error as evaluation metric for this model, this parameter was set to 'neg_mean_absolute_error'. After all the parameters were set, training was started with one extra parameter - number of iterations. Number of iterations was set to 700. After 700 iteration training with test data and cross validation data set, a benchmark model was born. In order to validate the benchmark model using neg mean absolute error, test dataset preparation was performed. Test dataset preparation process was the same as training dataset preparation. Test data sets was merged first. Then missing values were filled out and non-numerical data was transformed. When test data was ready, benchmark model was used to predict targeted values (logerror) for test dataset. 'neg_mean_absolute_error' parameter from skLearn[17], was used to get MAE score. Predicted targeted values and actual target values (logerror) were put into **mean_absolute_error**. NegMAE score for benchmark model is -0.05949. After benchmark model was tested out, couple improvements were made from this model and couple new model were implemented based on this model. The first model was made after benchmark model used the same parameters and techniques. The only difference is that this new model set different values for all this parameter after try and errors as well as tuning. The first new model was called lgb_model. **neg_mean_absolute_error** for lgb_model is-0.06966. Next,

model was made used different algorithm - XGBoost and different parameters. It's called xgb_model. **neg_mean_absolute_error** for xgb_model is -0.05317.

Process used in project to select the best model:

Step 1**:** Loading the crucial packages. Here, I am importing the required libraries.

```
#Import Required Libraries
import numpy as np # for efficient numerical computations
import pandas as pd # for data management
import matplotlib.pyplot as plt #to customize visualizations
import seaborn as sns #for easy/common visualizations

#Importing sklearn libraries
from sklearn.model_selection import cross_val_score
from sklearn.cross_validation import StratifiedKFold
from sklearn import cross_validation
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import cross_val_score
from sklearn import linear_model
from sklearn import tree
from sklearn import svm
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingRegressor

#Importing xgboost
from xgboost import XGBRegressor

import warnings
warnings.simplefilter("ignore")
```

```
# run model
xgb_params = {
    'eta' : 0.04,  # 0.037 grid search = .04
    'max_depth' : 6,  #5
    'subsample' : 0.80,
    'objective' : 'reg:linear',
     scoring: 'neg_mean_absolute_error'
    'lambda' : 0.8,
    'alpha' : 0.4,
    'base_score' : y_mean,
    'silent' : 1,
    'min_child_weight': 5  # grid search
}

dtrain = xgb.DMatrix(x_train, y_train)
dtest = xgb.DMatrix(x_test)

num_boost_rounds = 250

# training
model = xgb.train(dict(xgb_params, silent=1), dtrain, num_boost_round=num_boost_rounds)

# predict
xgb_pred1 = model.predict(dtest)
```

Parameters:

eta :

- Eta is used to shrink the size and it avoids overfitting.
- It ranges from 0 to 1.

max_depth

- Max depth of the tree. Overfit will happen if we try to increase this value.
- Ranges from 0 to ∞.

subsample

objective [default=reg:linear]

- reg:linear: linear regression

scoring

Regression	
'explained_variance'	metrics.explained_variance_score
'neg_mean_absolute_error'	metrics.mean_absolute_error
'neg_mean_squared_error'	metrics.mean_squared_error
'neg_mean_squared_log_error'	metrics.mean_squared_log_error
'neg_median_absolute_error'	metrics.median_absolute_error
'r2'	metrics.r2_score

The above screenshot shows that for regression problem I got use **neg_mean_absoulte_error.**

lambda [default=1, alias: reg_lambda]

alpha [default=0, alias: reg_alpha]

base_score [default=0.5]

silent [default=0]

min_child_weight [default=1]

Results

9.1 Random Forest Regressor

Random forest is a benchmark model and is popular in solving regression and classification problem. Random forest sums up all the decision tree to reduce the overfitting. A tree is traversed until a decision is reached and simple tests are performed on each tree.

I imported the RandomForestRegressor library. In order to evaluate the benchmark model using mean absolute error, test dataset preparation and train dataset was performed. Test data sets was merged first. Then missing values were filled out and non-numerical data was transformed. When test data was ready, benchmark model was used to predict targeted values (logerror) for test dataset. 'neg_mean_absolute_error' parameter from skLearn[17] was used to get neg_mean_absolute_error score. Predicted targeted values and actual target values (logerror) were put into neg_mean_absolute_error. All the parameters and

dataset where loaded into the RandomForestRegressor() function. After tuning the RandomForestRedressor() and with Cross Validation technique[Figure18] I achieved -**0.05949** average neg_mean_absloute_error score for benchmark model. cross_val_score is a cross validation value, which is utilized to get the k-folds. Cross validation technique is generally used to assess the predictive ability of any regression model. k-folds= 10 was used.

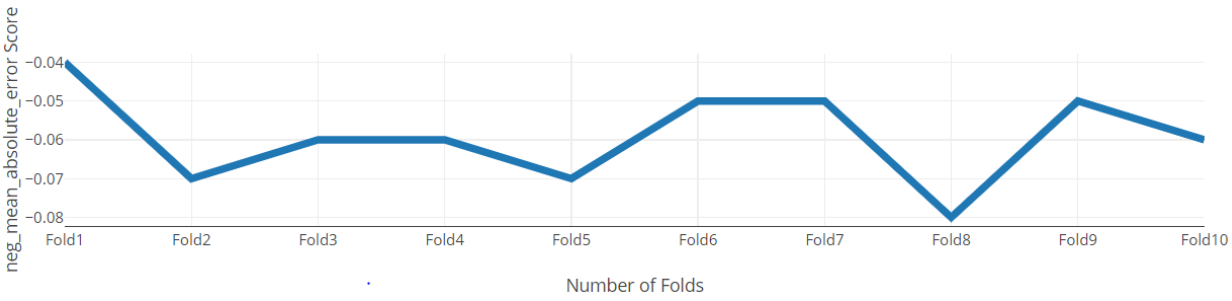

Figure 18: neg_mean_absolute_error score of RandomForestRegressor.

First, I used the random forest regressor to test the predictive algorithm with all the neg_mean_absolute_error score. I used a regression model with all the variables. Data preprocessing was done as explained above. Using k fold cross validation technique, I noticed the follow neg_mean_absoulte_error score. After analyzing this model 8 out of 10 folds shows high scores (Figure 18), which is 80% which is closer to 0. Finally, I got the average neg_mean_absloute_error score as **-0.05949**.

Below shows the code and the output for Random Forest.

```
Model: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
Score: -0.0594988842515
```

9.2 Gradient-Boosted Tree Regression

Gradient-boosted tree are ensembles of decision trees. This model uses booting technique to iteratively train the decision trees[18]. Each tree is dependent on the previous tree and

reduces the discrepancy by learning from the previous tree. The test dataset preparation and train dataset preparation were done same way and all the pre-processing steps were done as explained in Section 9.1. Next all the parameters and data were loaded into GradientBoostingRegressor() function. After tuning the GradientBoostingRegressor() and with Cross Validation technique[Figure 19] I achieved **-0.05309** average neg_mean_absolute_error score.

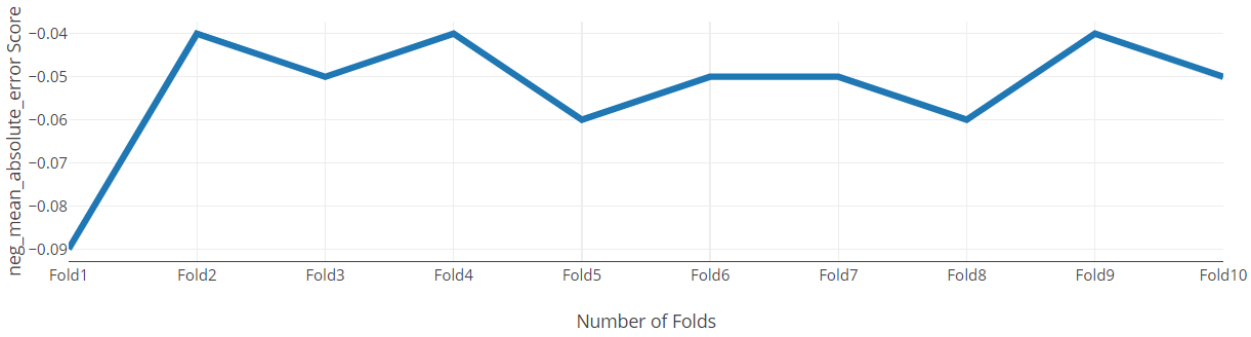

Figure 19: neg_mean_absolute_error score of GradientBoostingRegressor.

Second, Gradient Boosting Regressor was used to test the model with all the neg_mean_absolute_error score. I used a regression model with all the variables. Data preprocessing was done as explained in section9.1. Using k fold cross validation technique, I noticed the follow neg_mean_absoulte_error score. After analyzing this model 8 out of 10 folds shows high scores (Figure 19). Finally, I compared Gradient Boosting Regressor with Random Forest and found that average neg_mean_absloute_error score for Gradient Boosting is **-0.05309**, which is better than Random forest regressor.

```
Model: GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
             learning_rate=0.1, loss='ls', max_depth=3, max_features=None,
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=1,
             min_samples_split=2, min_weight_fraction_leaf=0.0,
             n_estimators=100, presort='auto', random_state=None,
             subsample=1.0, verbose=0, warm_start=False)
Score: -0.0530904245252
```

9.3 Bayesian Ridge Regression

The next model I worked on was Bayesian Ridge regression. Interface of Bayesian is used as a context to undertake the analysis of the statistical and work towards linear regression. After pre-processing the data, explained in section 9.1. The test and train dataset along with the parameters I loaded in to BayesianRidge () function. After tuning the BayesianRidge () and with Cross Validation technique[Figure 20] I achieved -0.05337 average neg_mean_absolute_error score.

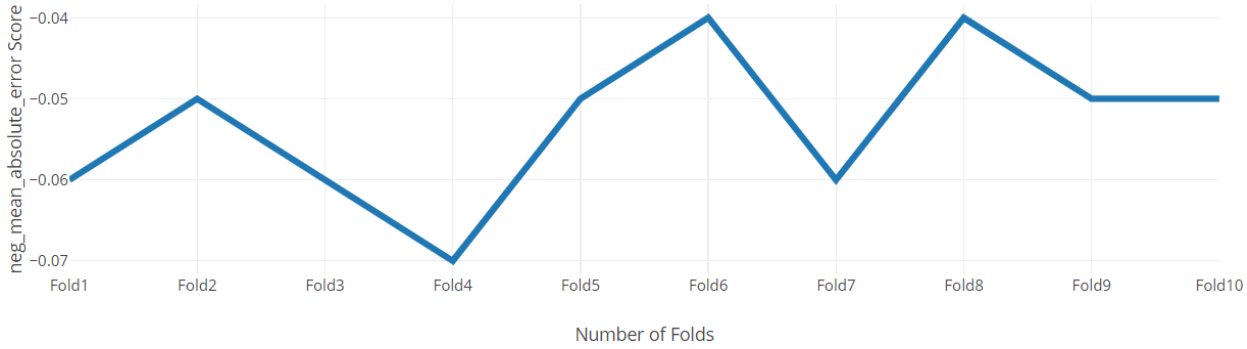

Figure 20: neg_mean_absolute_error score of BayesianRidge.

Third, Bayesian Ridge was used to test the predictive algorithm, with all the neg_mean_absolute_error score. I used a regression model with all the variables. Data preprocessing was done as explained in section 9.1. Using k fold cross validation technique, I noticed the follow neg_mean_absoulte_error score. After analyzing this model 8 out of 10 folds shows high scores (Figure 20). Finally, I compared Bayesian Ridge with Gradient Boosting Regressor and Random Forest and found that average neg_mean_absloute_error score for Bayesian Ridge is **-0.05337**, which is not a good compared to models

```
Model: BayesianRidge(alpha_1=1e-06, alpha_2=1e-06, compute_score=False, copy_X=True,
       fit_intercept=True, lambda_1=1e-06, lambda_2=1e-06, n_iter=300,
       normalize=False, tol=0.001, verbose=False)
Score: -0.0533737062301
```

9.4 Ridge Regression

Ridge Regression is an optimization of Ordinary Least Squares Regression. They are both linear regression models. The key difference is that these focus on regularization to prevent overfitting as the coefficient increases. As explained in section 9.1, the pre-processing of data was carried out. After the training and test the data. I loaded the data and parameters into Ridge() function. After tuning the Ridge () and with Cross Validation technique[Figure 21] I achieved -0.053385 average neg_mean_absolute_error score.

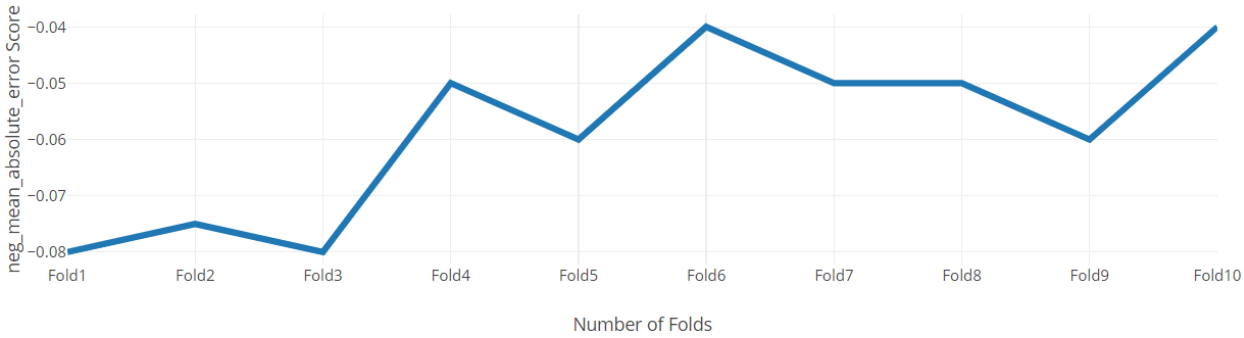

Figure 21: neg_mean_absolute_error score of Ridge Regression

Next, Ridge Regression was used to test the predictive algorithm, with all the neg_mean_absolute_error score. I used a regression model with all the variables. Data preprocessing was done as explained in section 9.1. Using k fold cross validation technique, I noticed the follow neg_mean_absoulte_error score. After analyzing this model 7 out of 10 folds shows high scores (Figure 21). Finally, I compared Ridge with Bayesian Ridge, Gradient Boosting Regressor and Random Forest and found that average neg_mean_absloute_error score for Bayesian Ridge is **-0.05338**, which is not a good compared to other models.

```
Model: Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
Score: -0.0533859935965
```

9.5 XGB Regression

XGBoost [19][20] algorithm was used for solving the problem. XGBoost is an implementation of gradient boosted decision tree and it stands as eXtreme Gradient

Boosting[Figure 8]. XgBoost is known for its flexibility, performance and speed. Compared to other model XGBoost is fast. This model is best used for tabular dataset and structured and it works best for classification and regression models. XGBoost is enabled with parallel processing making it at least ten times faster than any other tree-based models. It avoids overfitting by Regularization. The same process was used to test and train the data as explained in section 9.1. After tuning the XGBRegressor and with Cross Validation technique [Figure 22] I achieved **-0.053036** average neg_mean_absolute_error score.

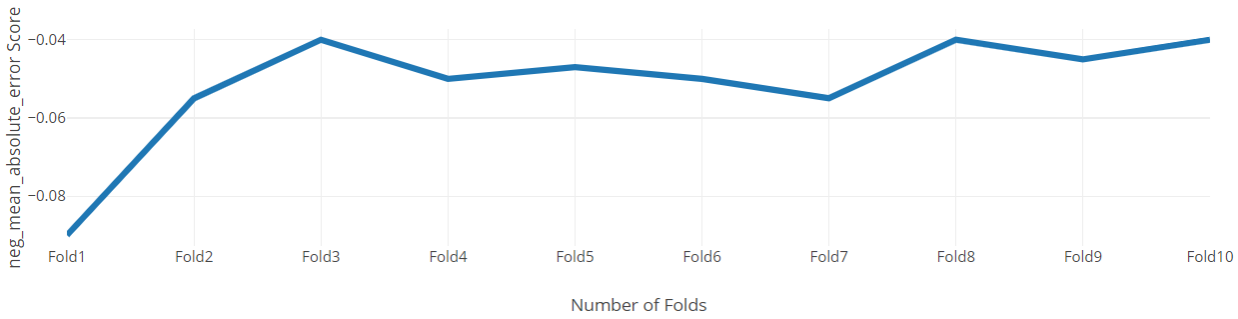

Figure 22: neg_mean_absolute_error score of XGBoost Regression

The final regression model, XGBoost Regression was used to test the predictive algorithm with all the neg_mean_absolute_error score. I used a regression model with all the variables. Data preprocessing was done as explained in section 9.1. Using k fold cross validation technique, I noticed the follow neg_mean_absoulte_error score. After analyzing this model 9 out of 10 folds shows high scores (Figure 22). Finally, I compared XGBoost with Ridge, Bayesian Ridge, Gradient Boosting Regressor and Random Forest and found that average neg_mean_absloute_error score for Bayesian Ridge is **-0.05303**, which performed better than rest of the regression model.

```
Model: XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
Score: -0.0530361127108
```

9.6 Refinement

The Benchmark Model RandomForestRegressor() gave us an neg_mean_absoulte_error to -0.05949. But the model XGBRegressor was chosen as it had a better score of -0.05303. Grid search is a technique through which we can come to know the best parameters for a machine learning model. It works in an iterative way. For some of the parameters associated with the model you enter some good probable values and the Grid search iterates through each of them and compares the result for each value and then gives you the best parameters which are suited for your model.

On GridSearch the score had a minor improvement of -0.05317. The parameter optimization was done on 3 tree-specific hyperparameters: 'n _estimators', 'max _depth' and 'min _child _weight'. Grid Search estimated the best parameters as: n _estimators = 100, max _depth = 3 and min _child _weight = 5. Using these parameters and upon cross validation the score improved to -0.05.

Result from Gridsearch

```
Best Score- -0.053170955805
Best Parameters- {'n_estimators': 100, 'max_depth': 3, 'min_child_weight': 5}
Best Estimator- XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=5, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

9.7 Model Evaluation and Validation

The final model was chosen after analyzing all major regression techniques, finalizing on using XGBoost, tuning the XGBoost using Grid Search and Cross Validation techniques. The robustness of the model and its solution were tested by using Cross Validation with number of folds as 10. k-fold cross-validation with k=10 was used for this schema. In this method, I train a model k times on (1-1/k)*100 percent of the training data (90% here). The validation was remaining 1/k*100 percent (10% here). The validation dataset is randomly selected from the training dataset[21]. Those k models are then used to predict for the test data, i.e. ~2.7M properties in the "properties" dataset[21]. The average values are then used to make

predictions. I used a 10-fold cross-validation (CV) method to train the model and make predictions. In this method, 10 equal size data are split in to the training data.

```
Cross validation again using best estimated parameters
Printing Final Score
MAE: 0.05 (+/- 0.00)
```

Figure 12: Cross validation using best estimated parameters

The following models were tried with default parameters.

```
Running XGB Regression
Running Linear Regression
Running Bayesian Ridge Regression
Running Ridge Regression
Running Lasso Regression
Running Decision Trees
Running Random Forest
Running KNN
Running Gradient Boosted Regressor
Returning the best model
Model: XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
Score: -0.0530361127108
Model: GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
             learning_rate=0.1, loss='ls', max_depth=3, max_features=None,
             max_leaf_nodes=None, min_impurity_decrease=0.0,
             min_impurity_split=None, min_samples_leaf=1,
             min_samples_split=2, min_weight_fraction_leaf=0.0,
             n_estimators=100, presort='auto', random_state=None,
             subsample=1.0, verbose=0, warm_start=False)
Score: -0.0530904245252
Model: Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
Score: -0.0533532241913
Model: BayesianRidge(alpha_1=1e-06, alpha_2=1e-06, compute_score=False, copy_X=True,
       fit_intercept=True, lambda_1=1e-06, lambda_2=1e-06, n_iter=300,
       normalize=False, tol=0.001, verbose=False)
Score: -0.0533737062301

Model: Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
Score: -0.0533859935965
Model: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
Score: -0.0533859948903
Model: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
Score: -0.0594988842515
Model: KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=5, p=2,
          weights='uniform')
Score: -0.0615386087447
Model: DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=None, splitter='best')
Score: -0.0846246402558
```

Sno	Model	neg_mean_absoulte_error Obtained
1	XGB Regression	-0.0530361127108
2	Linear Regression	-0.0533859948903
3	Bayesian Ridge Regression	-0.0533737062301
4	Ridge Regression	-0.0533859935965
5	Lasso Regression	-0.0533532241913
6	Decision tree	-0.0846246402558
7	Random Forest	-0.0594988842515
8	KNN	-0.0615386087447
9	Gradient Boosted Regressor	-0.0530904245252

Table 2: Comparing XGBoost with other models.

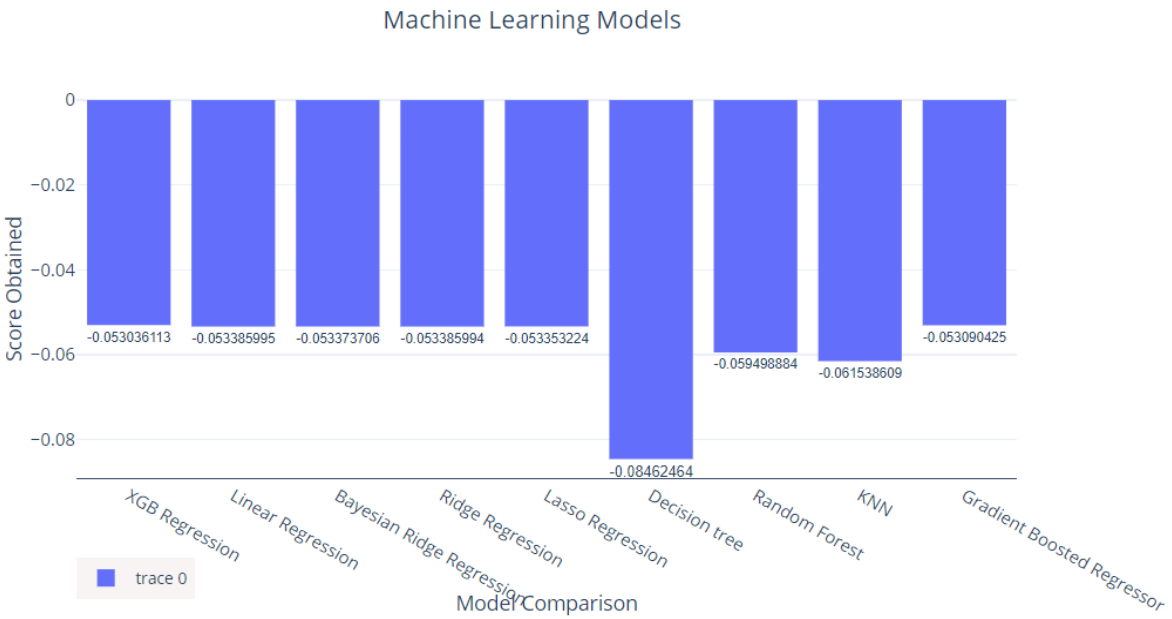

Figure 24: Comparing with other models with default parameters.

9.8 Justification

The final result of -0.05 generated by XGBoost is a very good compared to -0.05949 generated by benchmark model RandomForestRegressor. The final score has parameters of learning rate as 0. 1, max _ depth of 3, min _child _weight of 5 and n _estimators of 100. I strongly feel that the steps followed in getting the result add significance and validity to the final solution obtained.

9.9 Importance of Features

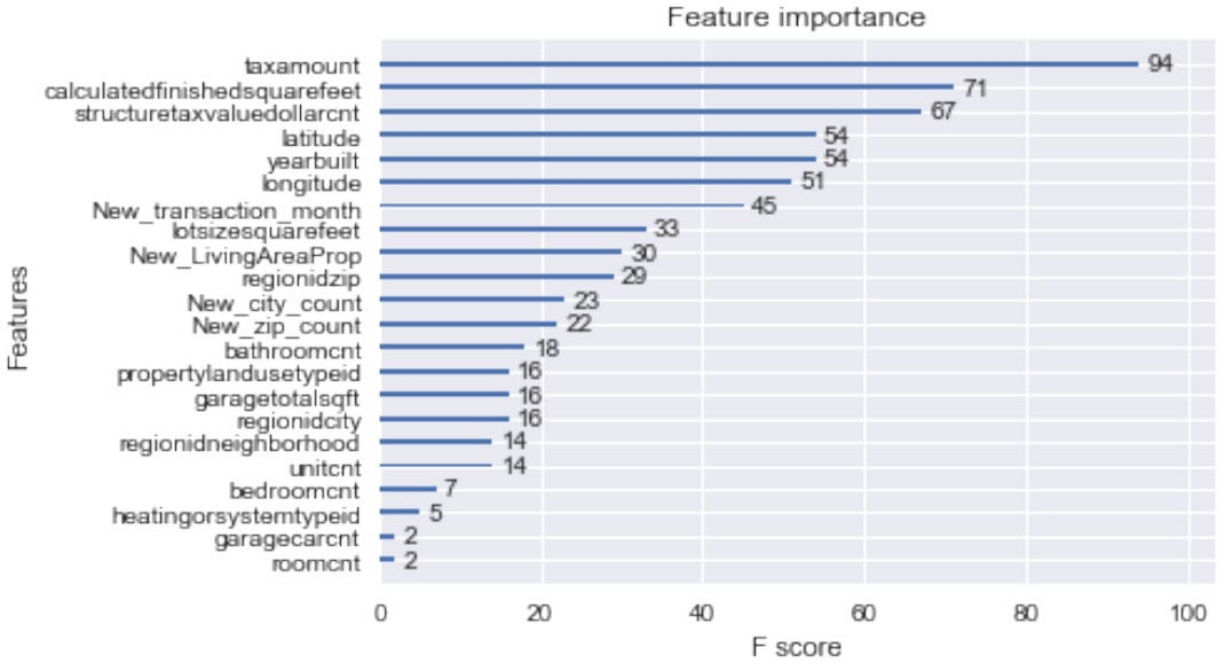

Figure 25: Showing F score obtained from using plot_importance feature of XGBoost

I used the plot_importance () built-in function provided by XGBoss to get the important features and this function maps features to corresponding F score. The above-mentioned diagram is obtained from using the plot_importance () feature of XGBoost. Based on domain knowledge of the Housing Industry I can state that the features generate by the model are in sync with the real-world housing market. For example, most of the home buyers are interested in paying low tax amounts, look for homes with high calculatedfinishedsquarefeet,

are interested in location of the house, age of the house - depicted by year built etc. All these features have been correctly marked as important by the model.

Conclusion

Purchasing a real estate property is often the highest and the important purchase that people make. People often spend hundreds of thousands and sometimes millions of dollars to buy a house or an apartment. When it comes to paying for the house, one needs to make sure that a fair sale price is placed on the property, and that the price of the house is not artificially inflated or shrank. Accurate estimation of home prices is extremely challenging because numerous factors influence the value of a property. These factors include neighborhood quality, size, tax, interest rate, and the overall economy. In the past, people have mostly relied on real estate agents to value their property. The human-operated evaluation of the properties had often led to large mismatches between the actual value of a property and the proposed sale prices.

Therefore, in this paper, I presented various features to consider while predicting the real estate properties. So, I have performed the preliminary assessment of the dataset like merging of data file and properties files, cleaning the data by dropping columns with more than 70% NaN values. I performed the correlation analysis to find the highly correlated features. Using K fold cross validation technique, I measured the performance of all models. I have used regression models. Compared all other models with XGBoost regressor, using various features to get the best neg_mean_absolute_error score. Overall, I found that for this particular problem, benchmark model Random Forest regression model is outperformed by XGBoost model.

Future Work

Some of the improvements I wanted to use by could not due to time and lack of knowhow, and think that the new model generated might be better than my model

I. Improve algorithm by using KNN to impute missing values
II. Implment LightGBM, Catboost algorithms and compare with XGBoost
III. Use CNN and ensemble it with LightGBM, Catboost algorithms and XGBoost

REFERENCE

[1] Kaggle dataset "https://www.kaggle.com/c/zillow-prize-1/data"

[2] R. J. Shiller, "Understanding recent trends in house prices and home ownership," National Bureau of Economic Research, Working Paper 13553, Oct. 2007. DOI: 10.3386/w13553. [Online]. Available: http://www.nber. org/papers/w13553.

[3] X. Yan and X. G. Su, Linear Regression Analysis. World Scientific Publishing, 2009.

[4] M. Bhuiyan and M. A. Hasan, "Waiting to be sold: Prediction of time-dependent house selling probability," in Proc. 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA), pp. 468–477, Oct 2016.

[5] W. T. Lim, L. Wang, Y. Wang, and Q. Chang, "Housing price prediction using neural networks," in Proc. 2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), 2016.

[6] Predicting Zillow Estimation Error Using Linear Regression and Gradient Boosting Darshan Sangani1, Kelby Erickson2, and Mohammad Al Hasan3

[7] D. C. Montgomery, E. A. Peck, and G. G. Vining, Introduction to Linear Regression Analysis. John Wiley & Sons, 5th ed., 2012.

[8] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," Annals of Statistics, vol. 29, pp. 1189–1232, 2000.

[9] J. H. Friedman, "Stochastic gradient boosting," Comput. Stat. Data Anal., vol. 38, pp. 367–378, Feb. 2002.

[10] L. Toscano, "Introduction to gradient-boosted trees and xgboost hyperparameters tuning (with python)," 2017.

[11] Breiman L (2001). "Random Forests". Machine Learning. 45(1): 5–32. doi:10.1023/A:1010933404324.

[12] A. Rogozhnikov, "Gradient boosting explained [demonstration]," 2016.

[13] Chen, Tianqi; Guestrin, Carlos (2016). "XGBoost: A Scalable Tree Boosting System". In Krishnapuram, Balaji; Shah, Mohak; Smola, Alexander J.; Aggarwal, Charu C.; Shen, Dou; Rastogi, Rajeev. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016. ACM. pp. 785–794. arXiv:1603.02754. doi:10.1145/2939672.2939785.

[14] "Story and Lessons behind the evolution of XGBoost". Retrieved 2016-08-01.

[15] Predicting Zillow Estimation Error Using Linear Regression and Gradient Boosting Darshan Sangani1, Kelby Erickson2, and Mohammad Al Hasan3

[16] Github link "https://github.com/scikit-learn/scikit-learn/issues/2439"

[17] D. Harrison and D. L. Rubinfeld, "Hedonic housing prices and the demand for clean air," Journal of Environmental Economics and Management, vol. 5, no.1,pp.81–102,1978.[Online].Available:http:// EconPapers.repec.org/RePEc:eee:jeeman:v:5:y:1978:i: 1:p:81-102.

[18] D. Belsley, E. Kuh, and R. Welsch, Regression Diagnostics: Identifying Influential Data and Source of Collinearity. New York: John Wiley, 1980.
[19] https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter
[20] J. R. Quinlan, "Combining instance-based and modelbased learning," Morgan Kaufmann, 1993, pp. 236–243.
[21] University of Washington "https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf"

Number of variables

	column Type	Count
1	int64	1
2	float64	53
3	datetime64[ns]	1
4	object	5