Capstone Project

Xiaochuan Yi

Machine Learning Engineer Nanodegree

May 18th, 2018

I. Definition

Porject Overview

Long term home price predication is always challenging. But short term home price predication is gaining success in recent years due to quality machine learning research done in this area. Housing data is publicly available including many features of houses and transaction price in market place. This makes home price predication is a suitable area to apply machine learning models. Companies such as Zillow and Redfin have already provided home price estimation services and their price estimation has become more and more valuable to customers, either pricing a home when selling or looking for offer price guidance when buying a home.

The Zestimate home valuation is Zillow's estimated market value, computed using a proprietary formula. It is not an appraisal. It is a starting point in determining a home's value. The Zestimate is calculated from public and user-submitted data, taking into account special features, location, and market conditions.

From my person experience, Zestimate is a very good guide to determine buying and selling price of a home in the metropolitan area where I live. I am curious of how Zestimate is calculated and why its precision is improving over the years. On May 24 2017, Kaggle opened a 1 Million dollar competition predicting short term future house price (Zestimate) residual errors. The participants were asked to develop an algorithm that makes predictions about the future sale prices of homes. At the time when I write the proposal, the competition is already over. However, the dataset of the competition is of high quality. In addition its scoring tool is still available for me to evaluate my solutions.

"Zestimates" are estimated home values based on 7.5 million statistical and machine learning models that analyze hundreds of data points on each property. And, by continually improving the median margin of error (from 14% at the onset to 5% today), Zillow has since become established as one of the largest, most trusted marketplaces for real estate information in the U.S. and a leading example of impactful machine learning.In this project, I created an Android application capable of reading aloud everyday text (e.g. product

Problem Statement

Following the Zillow competition, for each property (unique parcelid) along with it many features, a log error will be predicted for each specified time point. I will be predicting 6 timepoints: October 2016 (201610), November 2016 (201611), December 2016 (201612), October 2017 (201710), November 2017 (201711), and December 2017 (201712). According to the competition, the submission file should contain a header and have the following format:

ParcelId,201610,201611,201612,201710,201711,201712

10754147,0.1234,1.2234,-1.3012,1.4012,0.8642-3.1412

10759547,0,0,0,0,0,0

etc.

Note that the actual log errors are accurate to the 4th decimal places.

The log error is defined as

logerror=log(Zestimate)−log(SalePrice)

The Kaggle competition provides real estate data for three counties in and around Los Angeles, CA. Each observation of a property has 56 features; no additional features from outside data sources will be allowed in the analysis.

The data files are

1. 2016 Training data file with ParcelId, actual logerror = log(Zestimate) - log(SalesPrice) and transaction date. The 2016 properties file contains feature information for 90,725 properties. These two data file can be merged(joined) on ParcelId to provide the training data set with all features.
2. Prediction set with only the feature information for 2,985,217 properties.

Submissions will be scored based on the MAE (mean absolute error) across all predictions.

What models will be used? Since the training data set is a mixture of binary, categorical and continuous numerical features, it is suitable to try decision tree based models. We will start with scikit-learn ensemble ExtraTreeRegressor for feature importance, then try LightGBM and XGBoost models. According to the feature importance, missing values and intuition. Let's try to drop some features from the training data set and reduce the complexity of the models. We will see if dropping certain features from the training data set improves the score. Another aspect of data pre-processing is to remove the outliers. We will see if removing the outliers help improve the score as expected. We will filling missing values with various option and see if this could improve the model performance.

Metrics

Linear regression can be used as the initial reference models. In Zillow Kaggle competition, various teams apply different models to the data sets such as decision tree based models, support vector machines or neural networks. The competition web site provides a ranking tool and a public leaderboard that I plan to benchmark my results.

Housing price prediction (6 time points future of training data set) will be submitted to the competition web site's ranking tool. With the submission I can get the evaluation on MAE (Mean Absolute Error) score between the predicted log error and the actual log error and also a position in the public leaderboard.

MAE(logerror)

logerror=log(Zestimate)−log(SalePrice)

Two of the most common metrics used to measure accuracy for continuous variables are Mean Absolute Error (MAE) and Root mean squared error (RMSE).

Mean Absolute Error (MAE) measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight.

$$\text{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

RMSE is a quadratic scoring rule that also measures the average magnitude of the error. It's the square root of the average of squared differences between prediction and actual observation.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Both MAE and RMSE are negatively-oriented scores, which means lower values are better. RMSE has the benefit of penalizing large errors more so can be more appropriate in some cases, for example, if being off by 10 is more than twice as bad as being off by 5. But if being off by 10 is just twice as bad as being off by 5, then MAE is more appropriate. From an interpretation standpoint, MAE is clearly the winner. RMSE does not describe average error alone and has other implications that are more difficult to tease out and understand. On the other hand, one distinct advantage of RMSE over MAE is that RMSE avoids the use of taking the absolute value, which is undesirable in many mathematical calculations.

Since the target variable logerror is continuous variable, we assume errors are of linear scale and simple to understand, thus we choose MAE of the logerror as the metric. The log transformation reduces the skewness of the zestimate and sale price.

II. Analysis

Data Exploration

There are a total of 58 features in the property data file including the property id (parcelid). These data files are loaded into pandas dataframe and each feature data type, distribution and missing value issues are reviewed.

Let's consider drop features with extreme missingness, duplication, and zero variance. Variables with over 90% missingness and no feasible way to determine the correct value were dropped. If variables captured the same information, such as FIPS (Federal Information Processing Standard code) and Zip Code, we only kept one. Finally, variables with the same value across all observations were dropped as they would have had no impact on our model.

Here is the list of all features and their data types:

1. parcelid int64
2. pooltypeid10 float64
3. pooltypeid2 float64
4. pooltypeid7 float64
5. propertylandusetypeid float64
6. rawcensustractandblock float64
7. regionidcity float64
8. regionidcounty float64
9. regionidneighborhood float64

10. regionidzip float64
11. roomcnt float64
12. storytypeid float64
13. threequarterbathnbr float64
14. typeconstructiontypeid float64
15. unitcnt float64
16. yardbuildingsqft17 float64
17. yardbuildingsqft26 float64
18. yearbuilt float64
19. numberofstories float64
20. structuretaxvaluedollarcnt float64
21. taxvaluedollarcnt float64
22. assessmentyear float64
23. landtaxvaluedollarcnt float64
24. taxamount float64
25. taxdelinquencyyear float64
26. poolcnt float64
27. poolsizesum float64
28. longitude float64
29. airconditioningtypeid float64
30. architecturalstyletypeid float64
31. basementsqft float64
32. bathroomcnt float64
33. bedroomcnt float64
34. buildingclasstypeid float64
35. buildingqualitytypeid float64
36. calculatedbathnbr float64
37. decktypeid float64
38. finishedfloor1squarefeet float64
39. calculatedfinishedsquarefeet float64
40. lotsizesquarefeet float64
41. finishedsquarefeet13 float64
42. finishedsquarefeet12 float64
43. finishedsquarefeet50 float64
44. finishedsquarefeet6 float64
45. fips float64
46. fireplacecnt float64
47. fullbathcnt float64
48. garagecarcnt float64
49. garagetotalsqft float64
50. heatingorsystemtypeid float64
51. latitude float64
52. finishedsquarefeet15 float64
53. censustractandblock float64
54. propertyzoningdesc object
55. fireplaceflag object
56. propertycountylandusecode object
57. hashottuborspa object

58. taxdelinquencyflag object

Pandas dataframe data type (int64, float64 and object) only provides limited information about feature variables. Let's look deeper into these features.

We found that there re discrete features with significant ordering semantics, such as numerical identifiers and numerical coding for air condition type, architectural styles, and so on.

- parcelid: Unique identifier for parcels (lots). In other word, unique identification number for each property
- airconditioningtypeid: air conditioning types. It is possilbe certain air conditioning is of higher end than others. So transform the type to ordered numbers will be helpful.
- architecturalstyleid: architectural styles, it is possible one style is more luxurious than other styles. So if we can set ordering that is related to house price, then code the style in ordered number will be helpful.
- buildingclasstypeid: building class type
- buildingqualitytypeid: building quality type
- decktypeid: deck type
- heatingorsystemtypeid: Type of home heating system
- typeconstructiontypeid: What type of construction material was used to construct the home

There are also numerical features that are continuous or fine grain, such as

- basementsqft: basement area in square footage
- finishedfloor1squarefeet: Size of the finished living area on the first (entry) floor of the home
- calculatedfinishedsquarefeet: Calculated total finished living area of the home
- finishedsquarefeet6: Base unfinished and finished area
- finishedsquarefeet12: Finished living area
- finishedsquarefeet13: Perimeter living area
- finishedsquarefeet15: Total area
- finishedsquarefeet50: Size of the finished living area on the first (entry) floor of the home
- garagetotalsqft: Total number of square feet of all garages on lot including an attached garage
- lotsizesquarefeet: Area of the lot in square feet
- yardbuildingsqft17: Patio in yard
- yardbuildingsqft26: Storage shed/building in yard
- poolsizesum: Total square footage of all pools on property
- taxamount: The total property tax assessed for that assessment year
- taxvaluedollarcnt: The total tax assessed value of the parcel
- structuretaxvaluedollarcnt: The assessed value of the built structure on the parcel
- landtaxvaluedollarcnt: The assessed value of the land area of the parcel
- latitude: Latitude of the middle of the parcel multiplied by 10e6, location of the property
- longitude: Longitude of the middle of the parcel multiplied by 10e6, location of the property

There are also numerical features that are discrete with small set of values, and intuitively these features exhibits ordering that is related to property valuation, such as:

- roomcnt: Total number of rooms in the principal residence

- threequarterbathnbr: Number of 3/4 bathrooms in house (shower + sink + toilet)
- bathroomcnt: number of bathrooms
- bedroomcnt: number of bedrooms
- calculatedbathnbr: calculate bath number
- fireplacecnt : number of fireplaces
- fullbathcnt: number of full bathrooms (sink, shower + bathtub, and toilet)
- garagecarcnt: Total number of garages on the lot including an attached garage
- numberofstories: Number of stories or levels the home has
- poolcnt: Number of pools on the lot (if any)

Boolean features are also identified, such as:

- fireplaceflag: is a fireplace present in the house, True or missing(treated as False)
- hashottuborspa: Does the home have a hot tub or spa, true or missing(treated as False)
- taxdelinquencyflag: Property taxes for this parcel are past due as of 2015, Y or missing(treated as N), transformed to True or False

Other features:

fips: Federal Information Processing Standard code, will be not used

regionidcounty: County in which the property is located

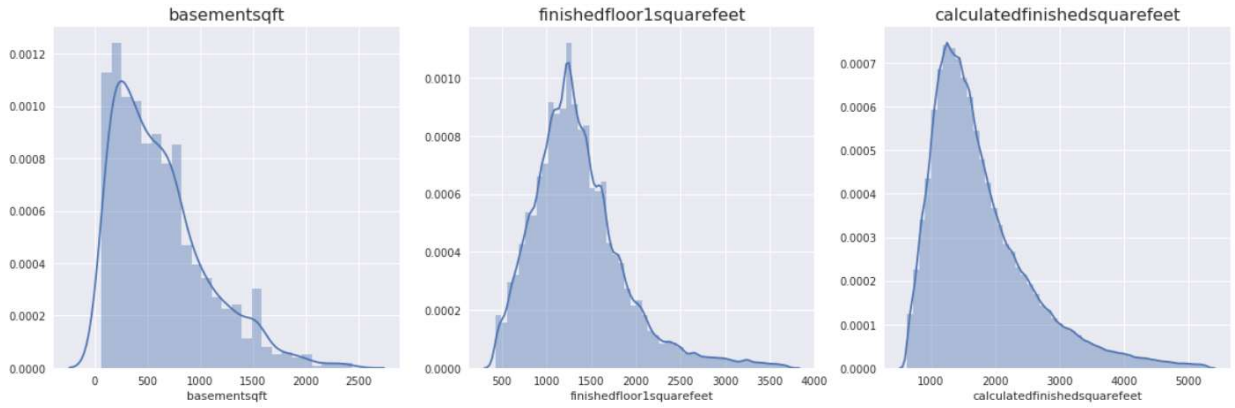
regionidcity: City in which the property is located (if any)

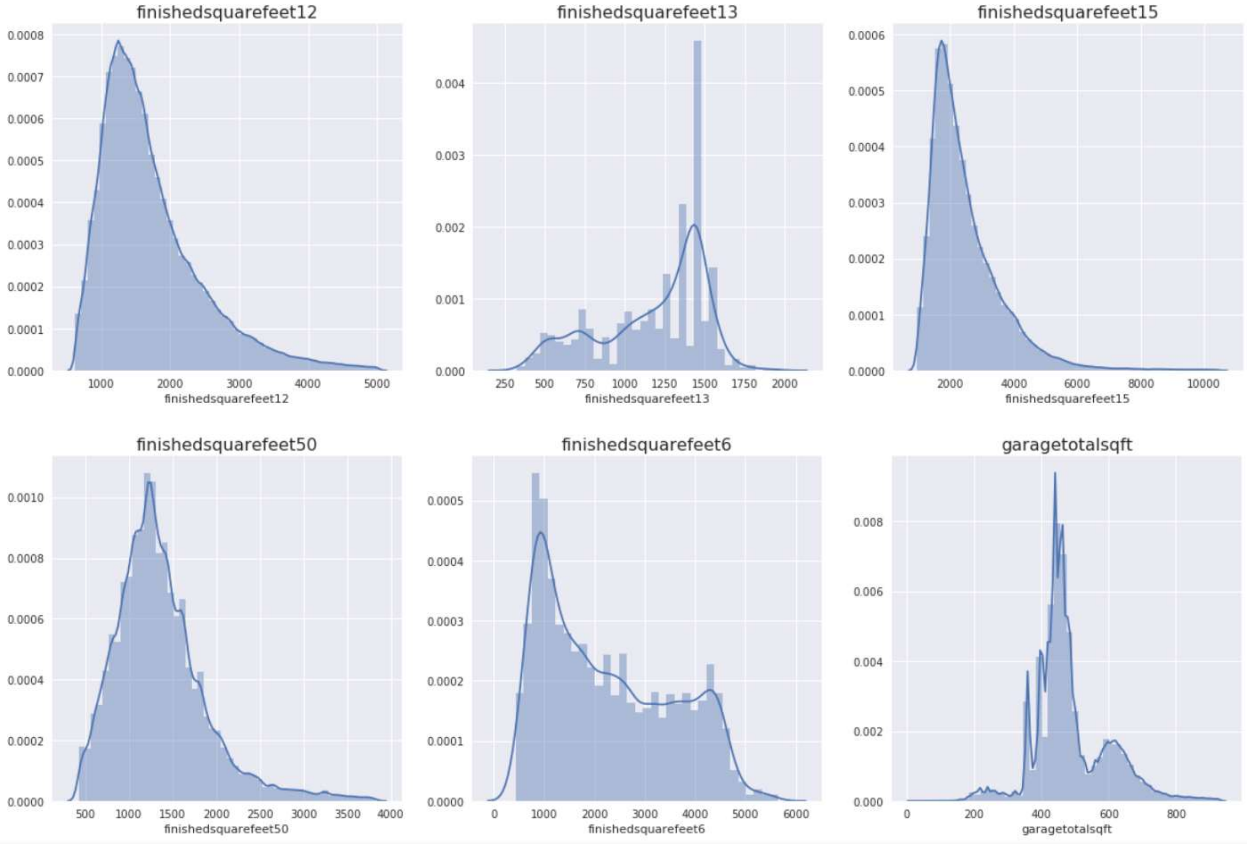
regionidzip: Zip code in which the property is located

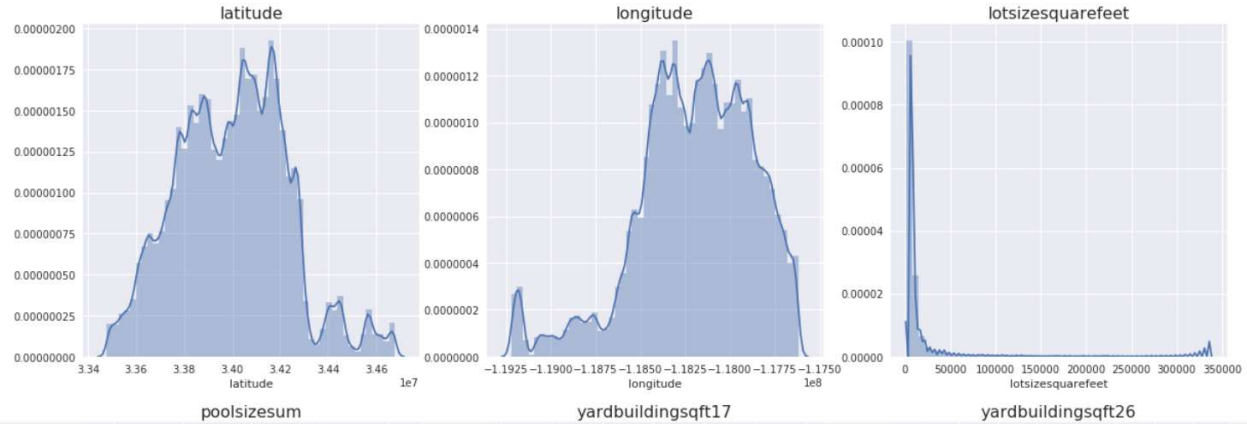
regionidneighborhood: Neighborhood in which the property is located

Exploratory Visualization

Now let's study the distribution of numerical data features with larger range. numerical_large_range = ['basementsqft', 'finishedfloor1squarefeet', 'calculatedfinishedsquarefeet', 'finishedsquarefeet12', 'finishedsquarefeet13', 'finishedsquarefeet15', 'finishedsquarefeet50', 'finishedsquarefeet6', 'garagetotalsqft', 'latitude', 'longitude', 'lotsizesquarefeet', 'poolsizesum', 'yardbuildingsqft17', 'yardbuildingsqft26', 'yearbuilt', 'structuretaxvaluedollarcnt', 'taxvaluedollarcnt', 'landtaxvaluedollarcnt', 'taxamount']. I have found that most of these features are skewed rather than symmetric or normal distributions. Check the following distribution plots.




finishedsquarefeet12
finishedsquarefeet12
finishedsquarefeet13
finishedsquarefeet13
finishedsquarefeet15
finishedsquarefeet15
finishedsquarefeet50
finishedsquarefeet50
finishedsquarefeet6
finishedsquarefeet6
garagetotalsqft
garagetotalsqft


latitude
latitude
1e7
longitude
longitude
1e8
lotsizesquarefeet
lotsizesquarefeet
poolsizesum
yardbuildingsqft17
yardbuildingsqft26

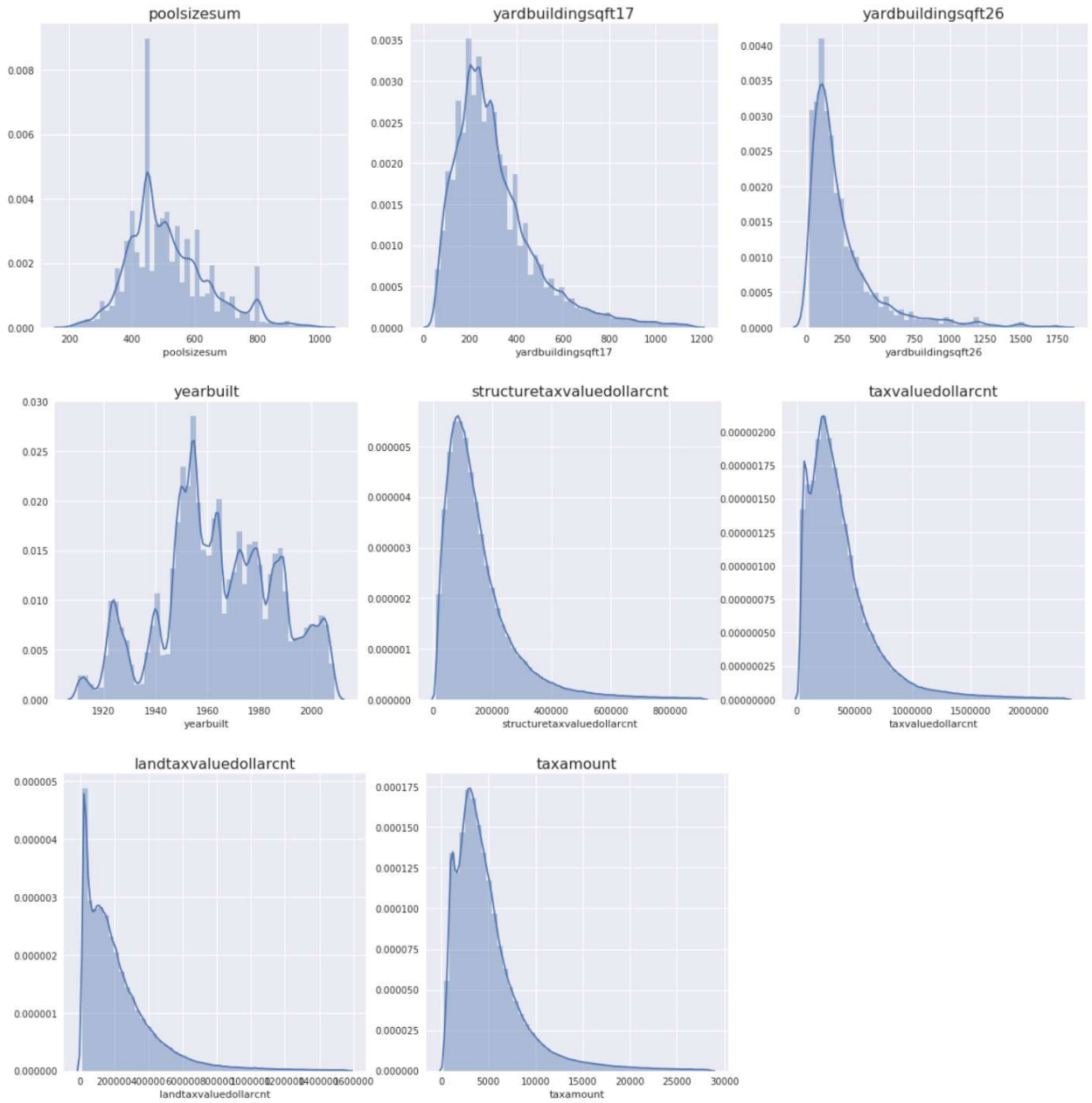
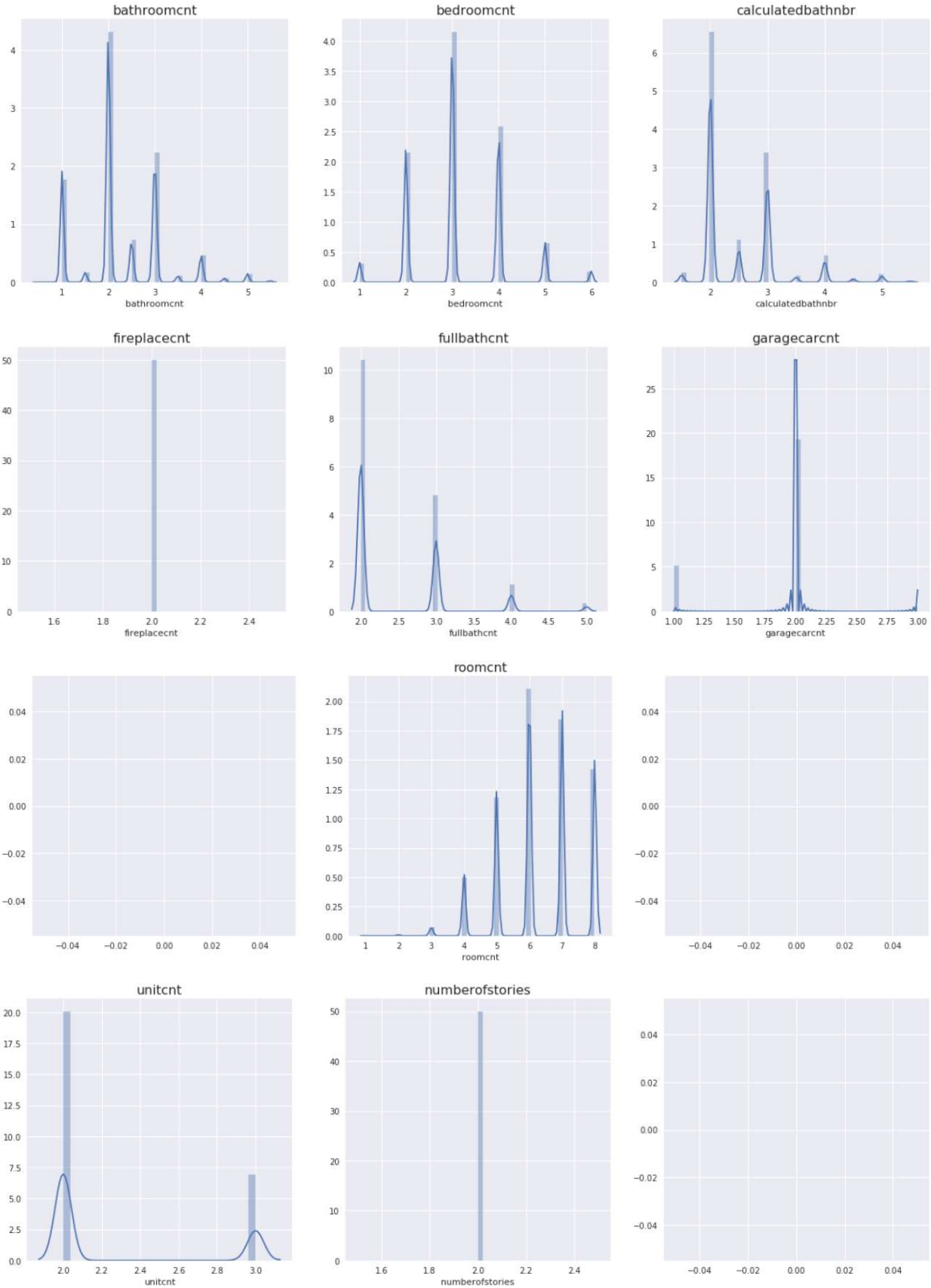


Figure 1. Distribution plots of numerical features that have larger range

Distribution of features that are discrete:


bathroomcnt
bathroomcnt
bedroomcnt
bedroomcnt
calculatedbathnbr
calculatedbathnbr
fireplacecnt
fireplacecnt
fullbathcnt
fullbathcnt
garagecarcnt
garagecarcnt
roomcnt
roomcnt
unitcnt
unitcnt
numberofstories
numberofstories

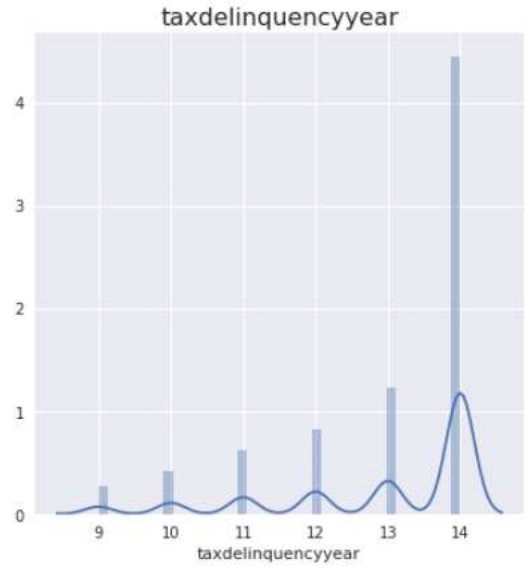


Figure 2. Distribution plots of discrete features

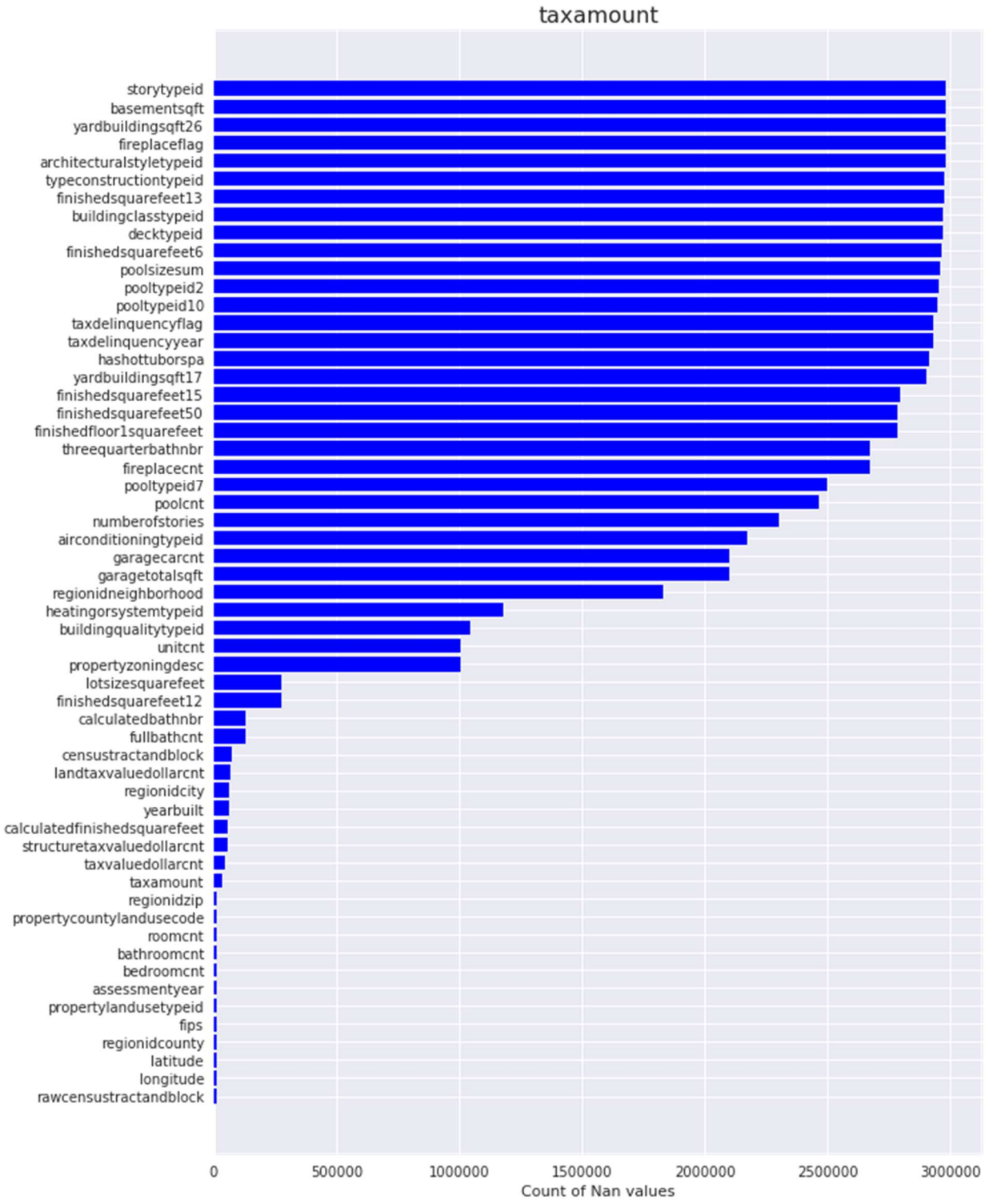


Figure 3. Count of missing values for each property feature.

Lastly, let's take a look at the target variable. The target variable is legerror, whcih has symmetric distribution with a mean value close to zero. This is not surprising since the log transformation has already been applied to Zestimate and sale price. Log transformation in a commonly used way to reduce the skewness of variables. Check the following distribution plot.

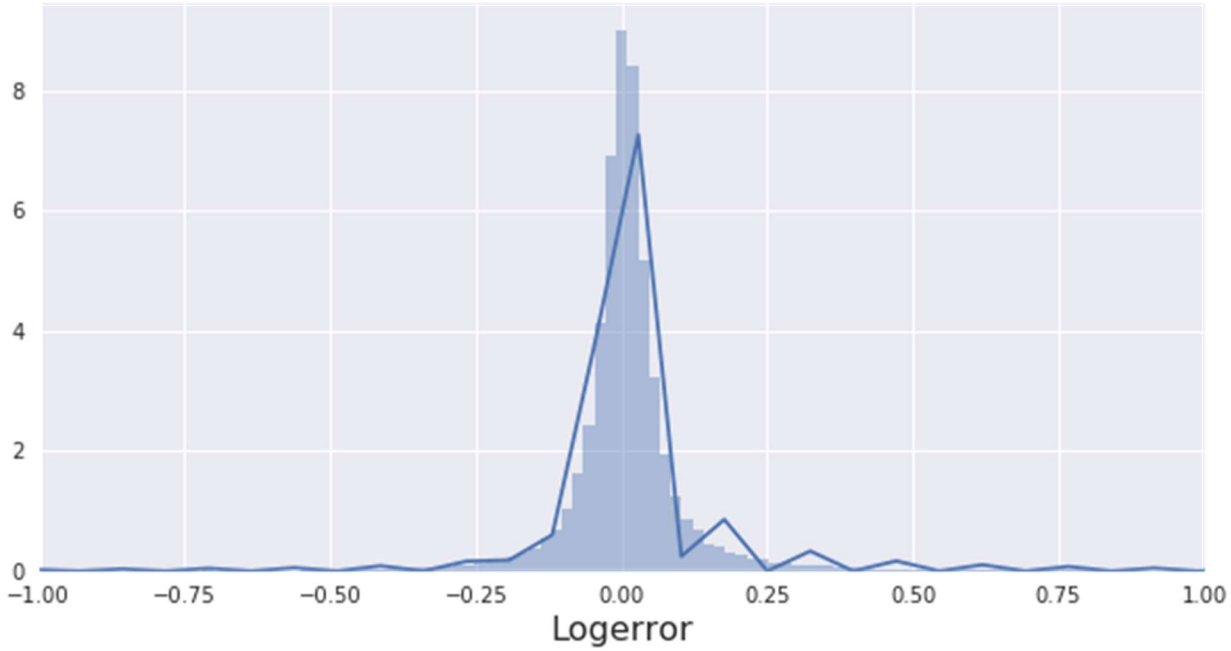


Figure 4. Distribution of target variable logerror

Algorithms and Techniques

After studying the data feature distribution and missing values, let's try techniques of feature selection and outlier removal to the training data set. A smaller set of features according to their importance can often reduce model complexity. In addition to selecting a smaller set of features, another key technique is to remove outliers of selected features. Outliers might have negative impact on machine learning performance.

Let's do a quick evaluation of feature importance using scikit-learn ensemble ExtraTreeRegressor.

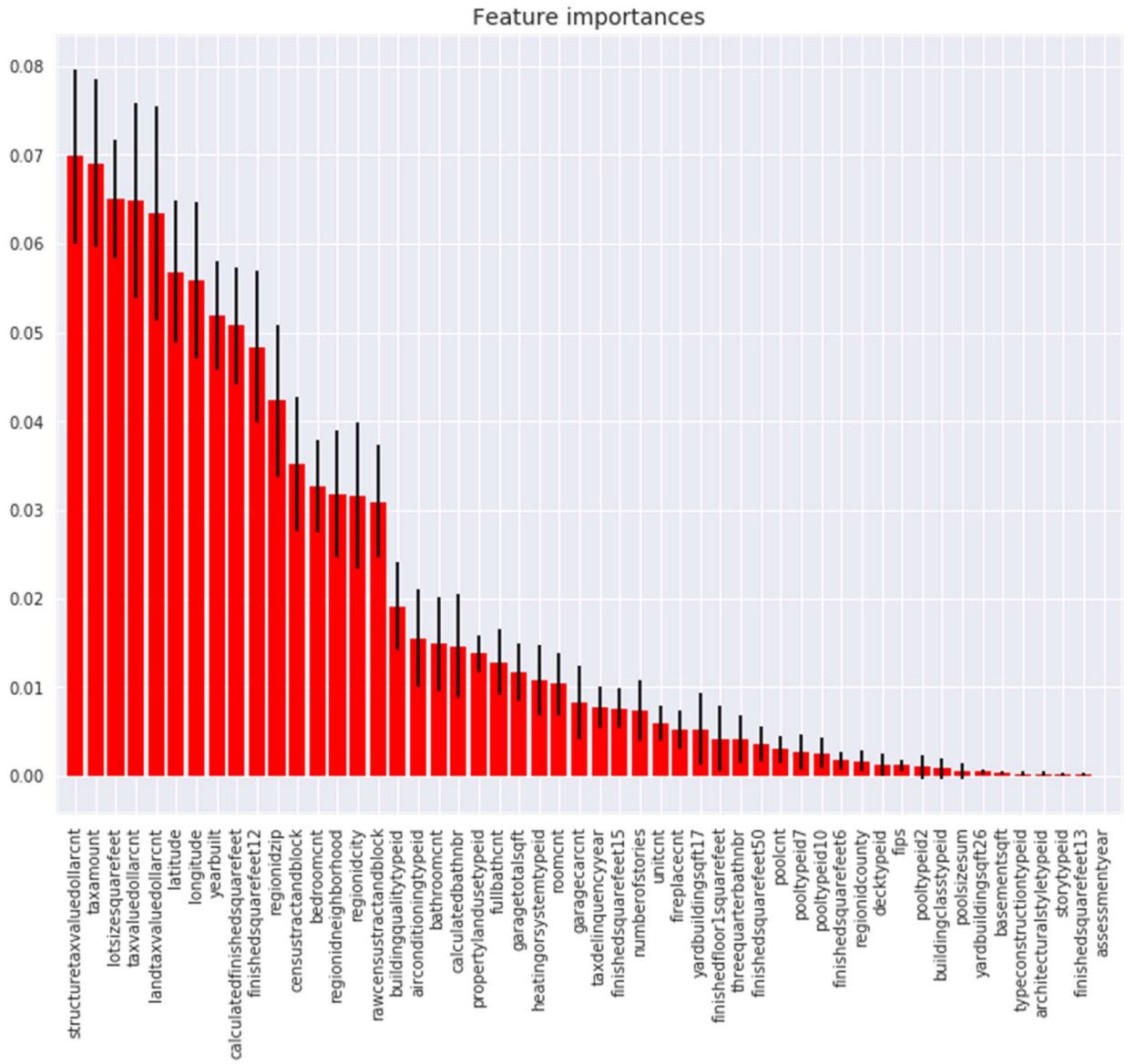


Figure 5. Feature Importance using ensemble.ExtraTreesRegressor

Since the training data set is a mixture of binary, categorical and continuous numerical features, it is suitable to try decision trees at first. Decision tree based algorithms result in model that can easily be visualized and understood by nonexperts (at least for smaller trees), and the algorithms are completely invariant to scaling of the features. The main downside of decision tree based algorithm is that even with the use of pre-pruning, they tend to overfit and provide poor generalization performance. Therefore, in most applications, the ensemble methods are used in place of a single decision tree, such as random forest. A random forest is essentially a collection of decision trees, where each tree is slightly different from the others. The idea behind random forests is that each tree might do a relatively good job of predicting, but will likely overfit on part of the data. If we build many trees, all of which work well and overfit in different ways, we can reduce the amount of overfitting by averaging their results. Random forests for regression and classification are currently

among the most widely used machine learning methods. They are very powerful, often work well without heavy tuning of the parameters, and don't require scaling of the data.

The gradient boosted regression tree(also called gradient boosting machines) is another ensemble method that combines multiple decision trees to create a more powerful model. Despite the "regression" in the name, these models can be used for regression and classification. In contrast to the random forest approach, gradient boosting works by building trees in a serial manner, where each tree tries to correct the mistakes of the previous one. By default, there is no randomization in gradient boosted regression trees; instead, strong pre-pruning is used. Gradient boosted trees often use very shallow trees, of depth one to five, which makes the model smaller in terms of memory and makes predictions faster. The main idea behind gradient boosting is to combine many simple models (in this context known as weak learners), like shallow trees. Each tree can only provide good predictions on part of the data, and so more and more trees are added to iteratively improve performance. Gradient boosted trees are frequently the winning entries in machine learning competitions, and are widely used in industry. They are generally a bit more sensitive to parameter settings than random forests, but can provide better accuracy if the parameters are set correctly. Gradient boosted decision trees are among the most powerful and widely used models for supervised learning. Their main drawback is that they require careful tuning of the parameters and may take a long time to train. Similarly to other tree-based models, the algorithm works well without scaling and on a mixture of binary and continuous features. As with other tree-based models, it also often does not work well on high-dimensional sparse data[4].

XGBoost (eXtreme Gradient Boosting) is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data. It was developed by Tianqi Chen and now is part of a wider collection of open-source libraries developed by the Distributed Machine Learning Community (DMLC). XGBoost is a scalable and accurate implementation of gradient boosting machines and it has proven to push the limits of computing power for boosted trees algorithms as it was built and developed for the sole purpose of model performance and computational speed. Because of its computation speed and model performance, it makes parameter tuning easier and improving performance further[5].

Based on proposal reviewer's recommendation of trying faster gradient boosting implementations with CatBoost or LightGBM, I tried LightGBM as follows. LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency
- Lower memory usage
- Better accuracy
- Parallel and GPU learning supported
- Capable of handling large-scale data

Benchmark

In Kaggle competition, there are many competitors running various models with the data set. Thus, I can benchmark my model using the evaluation scores with others.

Submit the data to Kaggle and we can get the evaluation on MAE (Mean Absolute Error) score between the predicted log error and the actual log error. In addition there is public leaderboard of 3779 contestants.

- MAE(logerror)
- logerror=log(Zestimate)−log(SalePrice)

The 60% percentile of ranked score is 0.0648409. I choose this score as the benchmark and plan to get inside the top 60% of the public leaderboard.

III. Methodology

Data Preprocessing

Let's run the pandas data analysis to find out the number of missing values for each feature. I have found that there are more than 33 features that have more than 25% missing values. I tried to drop all 33 features that have more than 25% missing values and it did not help with the score.

Then I tried to drop smaller number of features and fill missing values. For 3 boolean features, I will consider replace missing value with False. This is because I found that fireplaceflag only takes on True or Nan. The missing value probably means that no fireplace is found in the property. Let's treat hashottuborspa taxdelinquencyflag similarly. For numerical features, I tried to fill missing values with median or -1. I also tried to drop two sets of features from the training date sets. First iteration the set of features to drop is: cols_to_drop0 = ['parcelid', 'logerror', 'transactiondate', 'propertyzoningdesc','propertycountylandusecode', 'fireplacecnt', 'fireplaceflag']. The second iteration the set of features to drop is cols_to_drop1 = ['parcelid', 'logerror', 'transactiondate', 'propertyzoningdesc', 'propertycountylandusecode', 'fireplacecnt', 'fireplaceflag', 'fips','latitude','longitude','poolcnt', 'rawcensustractandblock', 'regionidcounty', 'regionidneighborhood', 'regionidzip']

Implemenation

LightGBM has been trained for 2 major iteration round. In first iteration, I have dropped the features in cols_to_drop0 = ['parcelid', 'logerror', 'transactiondate', 'propertyzoningdesc','propertycountylandusecode', 'fireplacecnt', 'fireplaceflag']. Missing values are filled with median. The first iteration produces submission_lightgmb.csv.gz with a score of 0.0650229, compared to the best competition score of 0.0631885 . This score ranks 2627th in the public leaderboard of 3779 participants.

In second iteration, I have dropped the features in cols_to_drop1 = ['parcelid', 'logerror', 'transactiondate', 'propertyzoningdesc', 'propertycountylandusecode', 'fireplacecnt', 'fireplaceflag', 'fips','latitude','longitude','poolcnt', 'rawcensustractandblock', 'regionidcounty', 'regionidneighborhood', 'regionidzip'] as fips is federal information standard processing code which is not related to housing valuation, similar to latitude and longitude are numerical values that indicates house location which is already indicated by zipcode. Feature poolcnt is also dropped and pool size is consider more related to house value. Again missing values are filled with median. The second run of LightGBM produces submission_lightgbm2.csv.gz with a score of 0.0647524 (2165th), which is an improvement from using more features in the first major iteration.

Finally XGBoost model has been trained with the feature set that used in the second LightGBM run. It achieves a score of 0.0645364 (rank of 1502nd, getting inside the first half of the learderboard). This is an improvement from previous result using LightGBM. However, LightGBM is faster to train and use less memory and it is very helpful with trying different feature selection and parameter configurations.

Refinement

Apply LightGBM

First let's take a look at the parameters of LightGBM that we plan to tune:

learning_rate: This determines the impact of each tree on the final outcome. GBM works by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in the estimates. Typical values: 0.1, 0.001, 0.003…

boosting: defines the type of algorithm you want to run, default=gdbt
gbdt: traditional Gradient Boosting Decision Tree
rf: random forest
dart: Dropouts meet Multiple Additive Regression Trees
goss: Gradient-based One-Side Sampling
num_boost_round: Number of boosting iterations, typically 100+
max_bin: it denotes the maximum number of bin that feature value will bucket in.
metric: again one of the important parameter as it specifies loss for model building. Below are few general losses for regression and classification.

- mae: mean absolute error
- mse: mean squared error
- binary_logloss: loss for binary classification
- multi_logloss: loss for multi classification

With the same training data, I tried the following 3 sets of LightGBM parameters and found that the second set of parameters deliver the best score. The second parameter set has max_bin = 32 compared to 10 in the first set of parameters. In the 3rd set of parameters, I tried to reduce number of leaves from 512 to 256, and min_data from 500 to 250, the resulting score is getting worse slightly.

```
params1 = {
    'boosting_type': 'gbdt',  #default=gbdt, type=enum, options=gbdt, rf, dart, goss, alias=boost,
    'objective': 'regression',
    'metric': 'l1',         # absolute loss, alias=mean_absolute_error, mae, regression_l1
    'verbose': 0,
    'max_bin': 10,
    'learning_rate': 0.0021, # shrinkage_rate
    'sub_feature':  0.5,     # feature_fraction
    'bagging_fraction': 0.85, # specifies the fraction of data to be used for each iteration and is generally used to speed up the training and avoid overfitting.
    'bagging_freq': 40,
    'num_leaves': 512,        # num_leaf
    'min_data': 500 ,         # min_data_in_leaf
    'min_hessian': 0.05     # min_sum_hessian_in_leaf
}
params2 = {
    'boosting_type': 'gbdt',
    'objective': 'regression',
    'metric': 'l1',         # or 'mae'
    'verbose': 0,

    'max_bin': 32,
        #max_bin, default=255, type=int
        #max number of bins that feature values will be bucketed in.
        #Small number of bins may reduce training accuracy but may increase general power (deal with over-fitting)

    'learning_rate': 0.0021,  # shrinkage_rate
    'sub_feature':  0.5,      # feature_fraction
```

```
    'bagging_fraction': 0.85, # specifies the fraction of data to be used for each iteration and is generally used to speed up the training and avoid overfitting.
    'bagging_freq': 40,
    'num_leaves': 512,        # number of leaves, default=31
    'min_data': 500,          #  default=20, alias min_data_in_leaf
    'min_hessian': 0.05       # min_sum_hessian_in_leaf,  default=1e-3
}
params3 = {
    'boosting_type': 'gbdt',
    'objective': 'regression',
    'metric': 'l1',         # or 'mae'
    'verbose': 0,

    'max_bin': 32,
        #max_bin, default=255, type=int
        #max number of bins that feature values will be bucketed in.
        #Small number of bins may reduce training accuracy but may increase general power (deal with over-fitting)

    'learning_rate': 0.0021,  # shrinkage_rate
    'sub_feature':  0.5,      # feature_fraction
    'bagging_fraction': 0.85, # specifies the fraction of data to be used for each iteration and is generally used to speed up the training and avoid overfitting.
    'bagging_freq': 40,
    'num_leaves': 256,        # number of leaves, default=31
    'min_data': 250,          #  default=20, alias min_data_in_leaf
    'min_hessian': 0.003      # min_sum_hessian_in_leaf,  default=1e-3
}
```

After the initial experiment with different parameters and getting different scores, let's take a more systematic approach to further the effort to find the best parameters for LightGBM – GridSearchCV.

Apply GridSearchCV to LightGBM

In [186]:
```
from sklearn.model_selection import GridSearchCV, GroupKFold
np.random.seed(1)
# other scikit-learn modules
regressor = lgb.LGBMRegressor(num_leaves=31)

param_grid = {
    'learning_rate': [0.001,0.0021,0.01],
    'n_estimators': [20, 40],
    'max_bin': [10,32,60],
    'boosting_type': ['gbdt','dart'],
    'objective': ['regression'],
    'metric': ['l1'],         # or 'mae'
    'verbose': [0],
    'sub_feature':  [0.5],      # feature_fraction
    'bagging_fraction': [0.85], # specifies the fraction of data to be used for each iteration and is generally used to speed up
    'bagging_freq': [40],
    'num_leaves': [256,512],        # number of leaves, default=31
    'min_data': [256,512],          #  default=20, alias min_data_in_leaf
    'min_hessian': [0.005,0.05]       # min_sum_hessian_in_leaf,  default=1e-3
}

model_gs = GridSearchCV(regressor, param_grid)
model_gs.fit(x_train, y_train)
print('Best parameters found by grid search are:', model_gs.best_params_)
```

```
('Best parameters found by grid search are:', {'num_leaves': 256, 'sub_feature': 0.5, 'min_hessian': 0.005, 'verbose': 0, 'bagg
ing_fraction': 0.85, 'learning_rate': 0.01, 'min_data': 256, 'n_estimators': 40, 'max_bin': 60, 'objective': 'regression', 'bag
ging_freq': 40, 'metric': 'l1', 'boosting_type': 'gbdt'})
```

The above parameter grid is the larger grid I have used. The size of the grid is 3x2x3x2x2x2 = 9 x 16 = 144. The advantage of LightGBM is its fast speed and low memory usage profile, which enables parameter running for large grids.

#score = 0.0648530, smaller grid,

#score = 0.0648389, larger grid, rank of 2273rd out of 3779, top 60% on the public leaderboard.

Apply XGBoost use GridSearchCV

XGBoost has also been applied and parameter tuning has been conducted with GridSearchCV. The resulting scores by XGBoost have turned out to be comparable or slightly worse. Since LightGBM has faster training speed and lower memory usage profile, it is chosen as the final model for the project.

Conclusion

The objective this project is to predict the log-error between Zillow Zestimate and the actual sale price, given all the features of a home. Firstly, I conducted exploratory analysis on the data set to find the insight into a data set, find outliers and anomalies, and to extract important variables. Then determined the machine learning model to apply. Secondly, I have chosen LightGBM as it is widely used in various machine learning models and it also handle Nan values that takes significant portion of the given data set. Thirdly, I have optimized the model parameters by applying grid-search cross-validation method and, also prevented overfitting the model to the training data by properly regularizing the model. I have also tried to apply XGBoost, but XGBoost method performance is slightly worse or similar to LightGBM. LightGBM is chosen as the final model as it has faster speed and lower memory usage profile. The proposed machine learning models work decently on the prediction of actual house selling price.

Key things learned

While I was working on this capstone project, I have learned many important techniques of applying machine learning algorithms to real world problem. First step is I have experienced the impotence of exploratory analysis, which helps me understand the data itself and choose suitable machine learning model for the problem on hand. I have also gained hands on experience of properly handling noisy data and its positive impact on the final model performance. This includes checking the integrity of the data and properly handling missing values. When we go on use the data sets to train the machine learning model, it is very important to optimize the parameters and properly regulate the model. Otherwise, the model might overfit to the training data so even the training error becomes very small, the actual error for unseen data may get significantly worse.

Future improvements

In this capstone project, the relationship between various features is not yet studied. With so many features in the training data set, many features are often related to each other. For example, bathroom count can be combined with bathroom size, and zip code is related to latitude and longitude. There are many different features related to the size of the house that can be merged to single data feature. There are many location related features that can be merge to a single data feature too. When the number of features can be significantly reduced, the noise in the data can be reduced, and the model complexity can be reduced and we will have higher probability to train a better model.

References

[1] Working with missing data in machine learning. https://towardsdatascience.com/working-with-missing-data-in-machine-learning-9c0a430df4ce

[2] Data type difference between categorical, ordinal and numerical features. https://stats.stackexchange.com/questions/312437/statistical-data-types-difference-between-categorical-ordinal-and-numerical

[3] https://stackoverflow.com/questions/45515031/how-to-remove-columns-with-too-many-missing-values-in-python

[4] Introduction to Machine Learning with Python: A Guide for Data Scientists

[5] XGBoost: A Scalable Tree Boosting System. https://arxiv.org/abs/1603.02754

[6] Light Gradient Boosting Machine. https://lightgbm.readthedocs.io/en/latest/