Zillow Home Value Prediction

by

Hadi Ramezani-Dakhel
November 2017

1. Definition

1.1. Project Overview

Buying a home is often the largest and the most important investment that people make during their lifetime. 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.

Because the majority of the data required for home value estimation is publicly available or can be obtained at low costs, computers and particularly machine learning algorithms can exploit these data to automate the valuation process. Zillow's Zestimate is a well-known framework for home value prediction using machine learning models. Zestimate was established more than a decade ago. Zestimate has since largely influenced the U.S. real estate market by predicting the price of millions of properties. Despite its tremendous success in home value prediction (with a median margin of error of 5% today [1]), there is still a lot of room for improvements (see ref. 2 to read about a recent lawsuit against Zillow). Introducing a reliable system for home value prediction will greatly help home buyers, particularly first-time buyers, receive a fair price for their properties.

Recently, Zillow has challenged the data science community to improve the accuracy of Zestimate through a $1,200,000 competition [1]. The competition has been administrated by Kaggle into two rounds: the qualifying round and the private round for the 100 top qualifying teams. In this project, I will illustrate how a predictive model can be developed to achieve a competitive score. Particularly, the strategies that we followed for an effective exploratory data analysis and features engineering are described in details.

1.2. Problem Statement

Zillow's Home Value Prediction competition is a two-round competition with an ultimate goal of improving the home-price prediction algorithm of the Zillow Company (aka Zestimate). During the first round of the competition, the objective is to predict the Zestimate's residual error:

$$logerror = log(Zestimate) - log(SalePrice)$$

This indicates that we need to predict where Zestimate fails and where it succeeds. To train a successful predictive model, our algorithm must be as good as Zillows' algorithm (not better and not worse). In the second stage, however, the objective is to actually improve the home value prediction algorithm. This project will focus on the first stage; my goal is to obtain a competitive average mean-absolute-error using feature engineering and model optimization.

This project is a supervised regression problem. Our goal here is to predict the residual error of Zestimate given available features and labels for ~90K data points. To solve this problem, one can use a regression algorithm such as Extreme Gradient Boosting, Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor, or linear regression among others. These algorithms attempt to best describe the actual value of the residual error given certain features. Once these models are trained, they can be used to predict the Zestimate's residual error of an unseen property. In this project, we will implement an Extreme Gradient Boosting model to predict the Zestimate's logerror. We will then improve our model by optimizing hyperparameters of the model along with extensive feature engineering. We show that our approach significantly improves the performance of a generic model to obtain a competitive score on the Kaggle's leaderboard.

1.3. Evaluation Metrics

The evaluation metric that will be used in this project includes mean-absolute-error (MAE). The mean-absolute-error corresponds to the expected value of the absolute error and is computed using the following formula [3]:

$$MAE(x^{true}, \hat{x}^{pred}) = \frac{1}{N_{samples}} \sum_{i=0}^{N_{samples}-1} \left| x_i^{true} - \hat{x}_i^{pred} \right|$$

Here, $N_{samples}$ is the number of samples, x_i^{true} is the true value of the i-th data, and $\hat{x}_i^{pred}$ is the predicted value of the i-th data.

Note that other evaluation metrics such as root-mean-square-error (RMSE) are valid metrics for this project. However, the Kaggle and Zillow teams have decided to make use of MAE in this competition. As such, we also use this metric to compute the score of our model.

2. Analysis

2.1. Data Exploration

In the Zillow Home Value competition, we are provided with information on ~3M “properties” located in three counties of California namely Los Angles, Orange, and Ventura. Figure 1 shows the timeline for this competition along with the available datasets. We are provided with two sets of training data and their corresponding “properties” file: (i) “train_2016_v2.csv” and "properties_2016", (ii) “train_2017.csv” and "properties_2017". The main difference between "properties_2016" and "properties_2017" are in tax features. The 2017 file contains updated tax values for the year 2017. Some other fields in 2017 file have also been updated. The file “train_2016_v2.csv” include the residual error information for all transactions that occurred from January 1, 2016 to September 30, 2016 and ~30% of transactions from October 1, 2016 to December 31, 2016. The remaining transactions are used to calculate the scores on the Kaggle’s public leaderboard. The file “train_2017.csv” similarly include 2017 transactions from January 1st through mid-September. The October-December, 2017 data will be used (once they become available) to compute the scores on the private leaderboard. These files can be obtained from the Kaggle website (https://www.kaggle.com/c/zillow-prize-1/data). In this project, we will focus on 2016 data but we will show that the feature engineering process is largely independent of the time period that the data was collected, and that it can seamlessly be applied to the 2017 data.

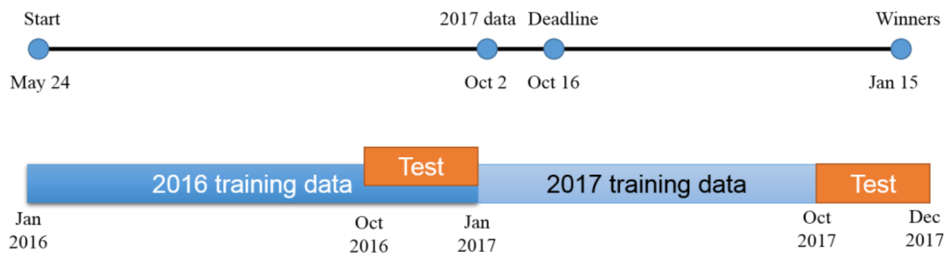


Figure 1. The timeline of the Zillow’s competition along with the training and test datasets. The competition has started on May 24, 2017. The deadline for the submissions was October 16, 2016. The winners will be announced on or around January 15, 2018. The scores on the public leaderboard is calculated using October-December 2016 transactions. The private leaderboard will be evaluated based on the October-December 2017 transactions once they become available.

2.2. Exploratory Visualizations

The data set "properties_2016" consists of 58 features and 2,985,217 observations. The features include information about the size, neighborhood, tax, and location of the properties. These features can be classified into 5 categories: (1) location, (2) area/size, (3) number, (4) quality, and (5) tax features (a small sample of this dataset is provided in Appendix A.1). Table 1 summarizes the features that are included in each category. In this project, we have renamed the features for easier interpretation of their contents [4].

Table 1. The primary features can be classified into 5 different categories. Each category contains between 7 and 15 features.

Location	Area/Size	Number	Quality	Tax
latitude	area_firstfloor_finished	num_bathroom	type_aircon	tax_building
longitude	area_total_calc	num_bedroom	type_architectural	tax_total
region_city	area_live_finished	num_bathroom_calc	type_framing	tax_year
region_county	area_liveperi_finished	num_fireplace	type_quality	tax_land
region_neighbor	area_total_finished	num_bath	type_deck	tax_property
region_zip	area_unknown	num_garage	type_heating	tax_delinquency
zoning_property	area_base	num_pool	pooltypeid10	tax_delinquency_year
zoning_landuse_county	area_basement	num_room	pooltypeid2	
fips	area_garage	num_75_bath	pooltypeid7	
censustractandblock	area_pool	num_unit	type_story	
rawcensustractandblock	area_lot	num_story	build_year	
	area_patio		type_material	
	area_shed		flag_tub	
			flag_firepalce	
			type_zoning_landuse	

We expect that the features within each category show some correlations. Figure 2 shows the correlation matrices for selected features in Table 1. Obviously, tax features are highly correlated. Additionally, a strong correlation between the area_total and number of beds and baths is observed.

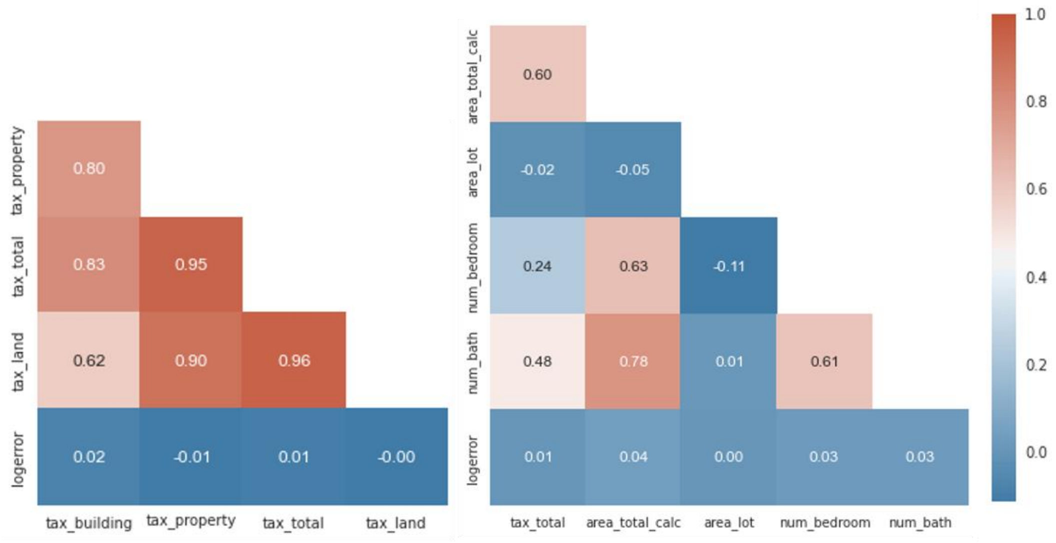
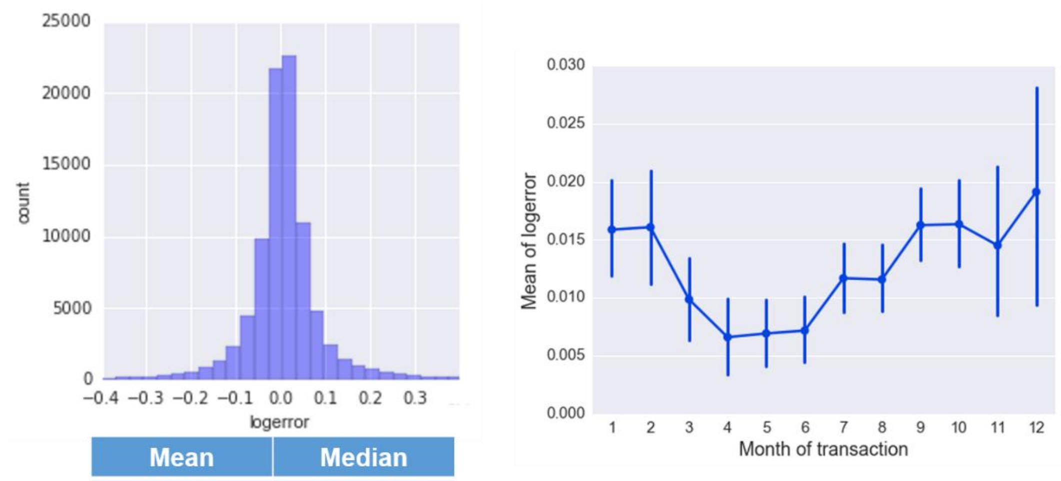


Figure 2. Correlation between tax and size-related features. Tax features are highly correlated. The correlations between the total_area and number of bed/bathroom is also strong.

The dataset "train_2016_v2.csv" contains 3 variables and 9,0275 observations. These variables include parcel_id, transaction date, and logerror (a small sample of this dataset is provided in Appendix A.2). Figure 2 shows the distribution of the logerror, and the change in the average logerror as a function of transaction month. The distribution of logerror is normal with a positive mean and median. This indicates that Zestimate, in general, tends to overestimate the actual sale price of the properties.



Mean	Median
0.011457	0.006000

Figure 2. Histogram of logerror (left) and the mean logerror as a function of transaction month (right).

The average logerror decreases as we enter the spring (March, April, etc) and it rises once the summer is over (September, October, etc). This may signal a general and possibly a periodic trend. However, the logerror information of several years is needed to confirm such behavior. As such, we decided to exclude any time-related information from our models.

For a comprehensive data analysis on this dataset visit my Github repository https://github.com/hadi-ramezani/ZillowPrize_EDA. I have also created an interactive animation using d3.js to show the distribution and the number of outliers in each month. This visualization can be found here: https://github.com/hadi-ramezani/D3_Visualization

2.3. Algorithms and Techniques

XGBoost is an ensemble method that makes predictions by combining the output of several trees. Unlike Random Forest, gradient boosting methods such as XGBoost build one tree at a time. Because the trees are built sequentially, XGBoost may take longer to build the trees. However, since the new trees use the information that was learned before, XGBoost trees are better learners. Similar to other tree-based methods, XGBoost is prone to overfitting but there are numerous strategies to avoid this problem. Because the ensemble tree methods generally use a subset of data for each tree and then combine the predictions of several trees, they are more robust than a single decision tree. The XGBoost library [6] is designed for speed and performance, and has shown to outperform other methods in Kaggle competitions. As such, we decided to use XGBoost in this project. Here, I present a brief overview of this method.

2.3.1. XGBoost: Extreme Gradient Boosting

XGBoost is a parallel gradient boosting algorithm [6]. Similar to other ensemble methods, the XBGoost regressor is composed of hundreds of weak learners. XGBoost optimizes an objective function. The objective function is a summation of a loss function and a regularization function. This function depends on the leaf weights, and the number of leaves in a tree. Because XGBoost uses second order Taylor expansion of the loss function, the second derivative of the loss function must be defined. XGBoost iteratively builds new trees using the information that it obtains from the previous steps. For more information see the tutorial here:

http://xgboost.readthedocs.io/en/latest/model.html

Several parameters can be tuned to optimize an XGBoost model. Below are the main parameters that was used to optimize our model in this project:

- learning_rate: this parameter scales down the steps along the gradient. It can be used to make the boosting process more conservative. The optimum value was found to be 0.01 here.
- max_depth: the maximum depth of each tree. The optimized value is 9 here.
- num_round: the number of rounds for boosting. We used 470 in this project.
- subsample: the ratio of the training data points that are randomly selected to construct each tree. We used 1 for this parameter.
- min_child_weight: the minimum number of samples that must be present in each node. Here we used 70.
- colsample_bytree: the ratio of the features that is used to construct a tree. The value of 0.5 was shown to return the best results here.

A grid search was used to optimize the XGBoost algorithm in this project. Note that the optimization of hyperparameters is an iterative process. This indicates that our model may need to be re-optimized once new features are added.

2.4. Benchmark

This project, by definition, is about minimizing the difference between predictions of our model and the existing Zestimate model. Zestimate itself is our benchmark model here. Accordingly, a small score would imply that our model compares well with our benchmark model and a large score would imply otherwise.

Because the scores on the Kaggle's public leaderboard are computed using the October-December 2016 data, the Kaggle team has provided a sample submission benchmark by using the average value of all data. The mean-absolute-error for this submission is 0.0663010 (see https://www.kaggle.com/c/zillow-prize-1/leaderboard). This score can also be used as a benchmark in this project.

3. Methodology

3.1. Data Preprocessing

Before fitting a model to our 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% missing values. Similarly, some "quality" feature had >90% missing values (Figure 3).

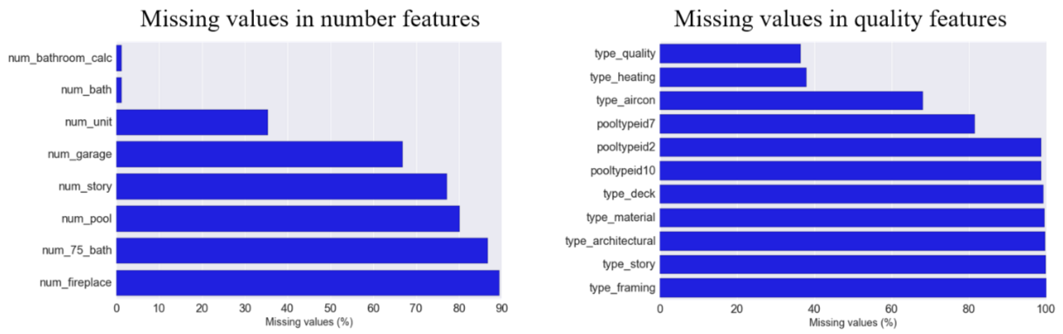


Figure 3. Percentage of missing values in "number" and "quality" features. Some features have up to 100% missing values. In this project, the missing "number" features were replaced by zero while the missing values of the categorical features were grouped into a new category.

In this project, we used zero to impute the missing "number" features. We note that imputing the missing "number" features with other values such as the "mean" or "median" did not improve the score of our model. This is because using "mean" or "median" here introduces some impurity to the data that eventually hurts the predictions of the algorithm. In contrast, imputing the missing values with zero, aggregates those data points together, and allows the tree-based methods to separate them from the rest of the data points. The missing values of the categorical ("quality") features were grouped together into a new category.

We followed a different strategy to impute the missing values in some of the location features. For those features, we used the coordinates (longitude and latitude) of a property to replace the missing values. The problem then turned into a k-nearest neighbor classification problem. For a property with a missing location feature, we identified the nearest 100 neighbors on the map to determine the missing location value. The value of the missing feature is obtained by voting among those neighbors (Figure 4).

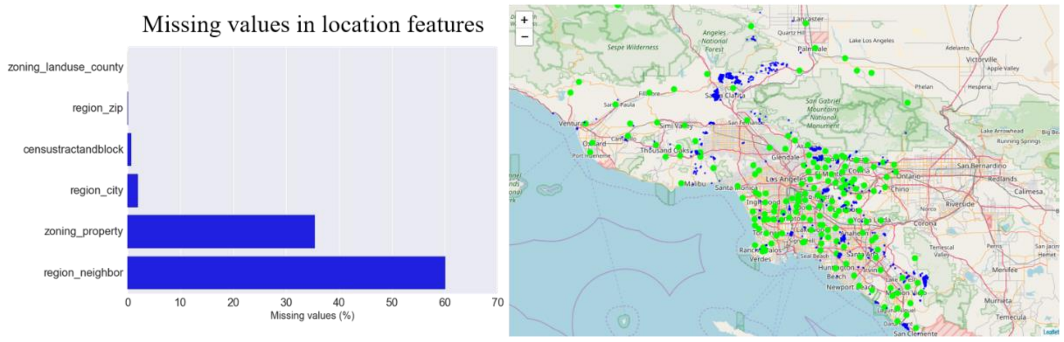


Figure 4. The missing values in location features. A k-nearest neighbor classification was used to impute the missing values here.

3.2. Implementation

The implementation process in this project involved an iteration between two main steps:

1- Feature engineering
2- Model training and testing

We used Jupyter notebooks and python scripts to complete the implementation. 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.

To train a model, we first read the data from the "properties" dataset. Then, we added new features to this dataset. Next, we read the "training" data and merged it with our augmented "properties" dataset. Last, we split the data into the training and validation sets using the methods explained in the following section to fit the model. To compute the mean-absolute-error (MAE), we used the mean_absolute_error method from the sklearn.metrics module. To implement the k-fold cross-validation scheme we used the KFold method from the sklearn.model_selection module.

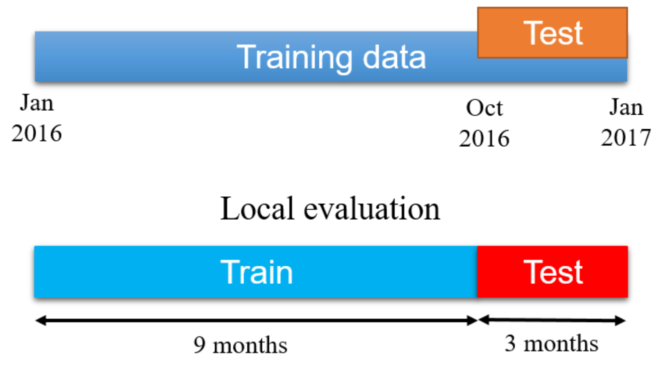
While the coding process was straightforward, computing and optimizing some of the new features required large amount of memory and several hours on a regular machine. We adopted a few strategies to alleviate these difficulties. For example, we split the "properties" dataframe into smaller chunks when making predictions (see the function "splitDataFrameIntoSmaller" in "xgboost.ipynb") to reduce the required memory. The codes that were used for feature engineering and training the models are provided in "codes" folder.

A generic implementation of XGBoost algorithm using the parameters: learning_rate = 0.02, max_depth = 7, subsample = 0.9, colsample_bytree = 1.0, num_roud = 240, led to a score of 0.0645278.

3.2.1. Cross-Validation

In Kaggle competitions, each team is limited to 5 submissions per day. Therefore, one needs to create a local cross-validation system that corresponds well to the computed score on the leaderboard. In this project, we decided to use two validation schemes:

1) Because Kaggle has reserved nearly 2/3 of the logerror data for the time-period October-December 2016 to compute the public leaderboard, we used the remaining data for that time-period for local validation of our models. We used logerror data of January-September 2016 to train our model, and reserved the October-December data to locally evaluate the trained models and to optimize the training algorithms (Figure 5).



LB score	3Month score
64.5248	64.9093

Figure 5. Two-third of the Zillow's logerror data during October-December 2016 is available to us, and the remaining data is used to compute the scores on the public leaderboard. We used this data to locally validate our model. This data was also used to optimize the hyperparameter of our models.

2) We used the k-fold cross-validation scheme with k=10. In k-fold cross-validation method, we train a model k times on (1-1/k)*100 percent of the training data (90% here). The remaining 1/k*100 percent (10% here) of the data is used for validation. The validation dataset is randomly selected from the training dataset. Those k models are then used to make a prediction for the test dataset, i.e. the ~2.7M properties in the "properties" dataset. The average values is then used to make predictions.

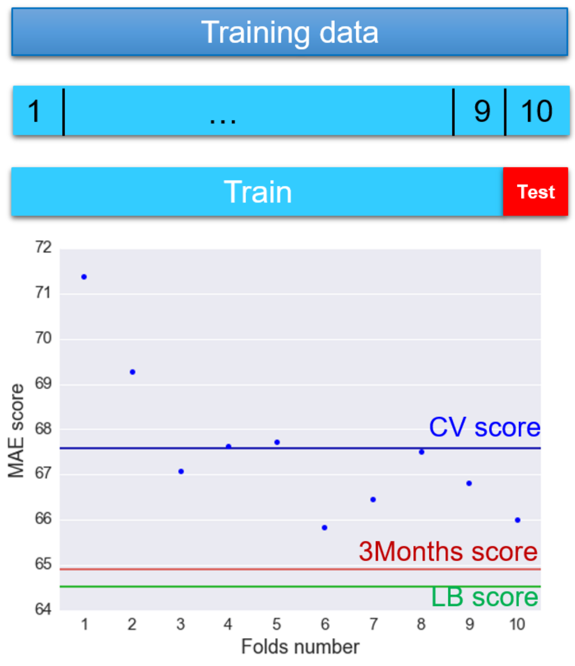


Figure 6. We used a 10-fold cross-validation (CV) method to train our model and make predictions. In this method, the training dataset is split into 10 equally-sized subset. The model is trained on 9 folds while the remaining fold is used for validation. Both 3month score and CV scores reasonably tracked the public leaderboard.

Note that a property can be sold multiple times during a certain time-period. This indicates that a property in the training dataset may be sold again, and the logerror for that property could be different depending on the actual sale price. Therefore, we are required to submit predictions for the ~90K properties in the training dataset as well. To do this, we excluded the training data from the properties dataset, and instead used the validation values obtained from our cross-validation scheme to make predictions for the future logerror values (Figure 7). This method helps prevent information leak if one decides to combine multiple methods that would potentially outperform the predictions of a single method.

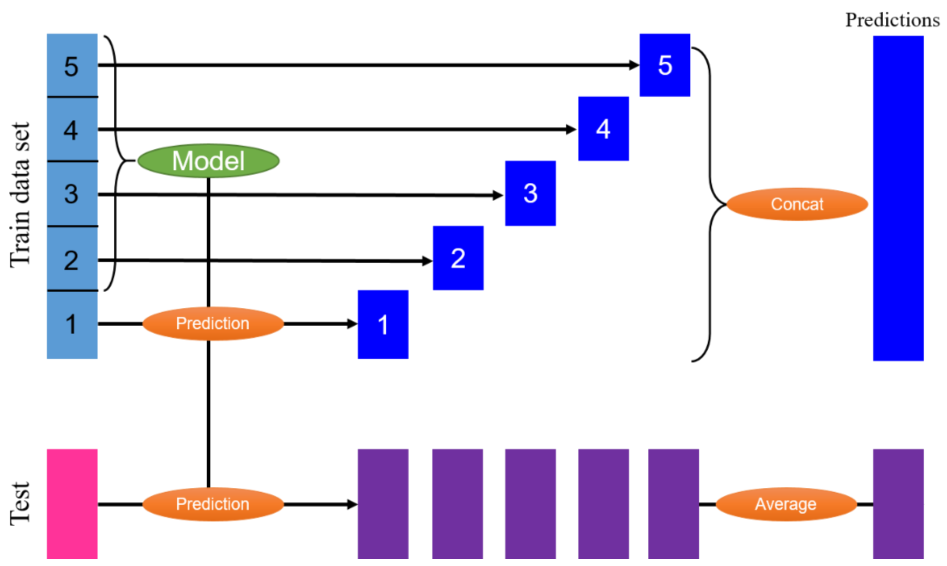


Figure 7. To make predictions for the future sale of the training dataset, we used the predictions on the validation sets during the cross-validation process. Because we used a 10-fold cross-validation method here, our model is trained 10 times. Each model is trained on 90% of the training data to make predictions for the remaining 1/k portion of the data as well as all of the properties in the "properties" data set. To submit a prediction, we concatenated the predictions on the training dataset, and averaged the 10 predictions made for the "properties" dataset.

3.4. Refinement

3.4.1. Outlier Detection

To develop a competitive predictive model in this project, it is crucial to properly recognize and eliminate outliers. Here we optimized an upper and a lower bound for the outliers' removal using the equation below:

$$Q1 - fac \times IQR < logerror < Q3 + fac \times IQR$$

Q1 and Q3 are the first and the third quartiles respectively. IQR is interquartile range (Q3- Q1) and fac is a constant. Outliers are generally defined as observations that fall below Q1 − 1.5 × IQR or above Q3 + 1.5 × IQR [6]. Because the distribution of the logerror is relatively narrow, we need to augment the value of the IQR multiplier to improve the score. Here, fac=8 is shown to return the best score (Figure 8).

Min	Q1	Median (mean)	Q3	Max
-4.605	-0.0253	0.006000 (0.011457)	0.0392	4.737

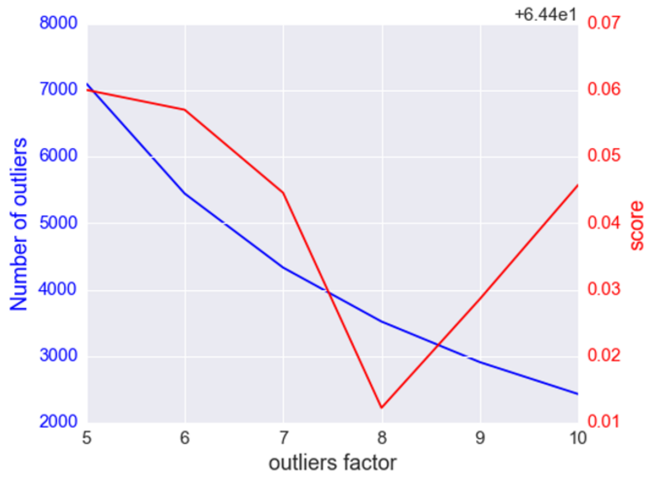


Figure 8. The optimum cutoff value for removing the outliers was obtained via fitting a model using different cutoff values. The cutoff range was calculated using the equation "fac × IQR". The variable "fac" was systematically varied between 5 and 10. IQR is the interquartile range (see text for more details)

Figure 9 shows the training and the evaluation procedure that we followed after dropping the outliers. In addition to computing the scores (using validation data) without outliers, we also used an arbitrary fac of 65 to compute a score that includes outliers in the predictions. The CV score obtained by dropping the outliers closely tracks the LB score on Kaggle’s website.

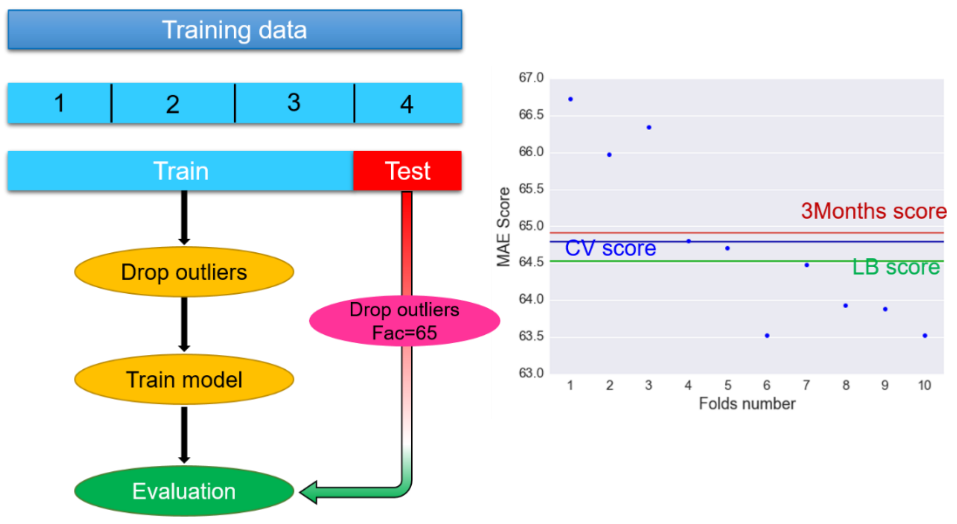


Figure 9. The procedure for model training and evaluation after dropping the outliers. The 3months and the CV scores corresponds well to the public leaderboard score on the Kaggle’s website. We also used an arbitrary fac value of 65 to compute the score with outliers present in the validation set.

3.4.2. Feature Engineering

Feature engineering is the process of using domain knowledge to create features that improve the performance of prediction models [8]. Feature engineering is exceptionally important in making successful predictive models. Before creating new features in this project, we fitted an XGBoost model to the training data using available features and the cutoff values described in the previous section. The computed score prior to the feature engineering was 0.0644122. We used this value as our benchmark to evaluate the effect of new features on the score. We also obtained feature importance data for our XGBoost model. This helps make informed decisions when creating new features (Figure 10). In this project, we engineered several new features as follows.

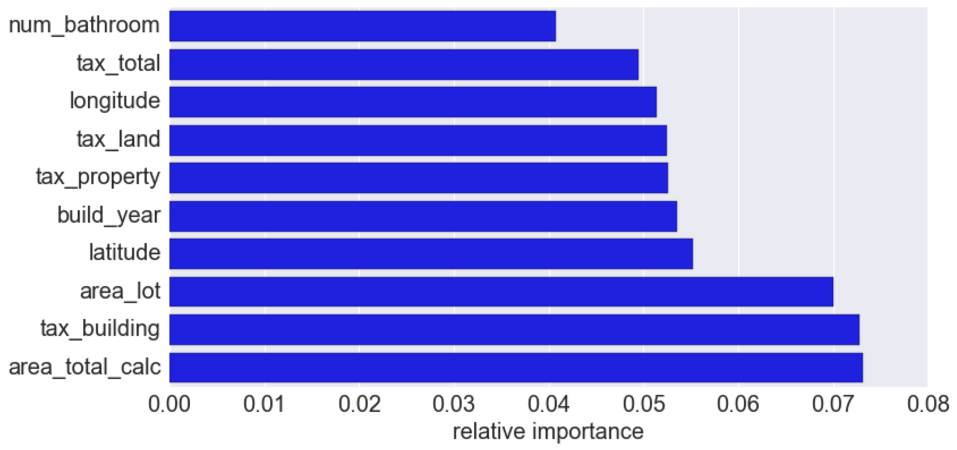


Figure 10. Relative feature importance obtained from a generic XGBoost model prior to feature engineering.

3.4.2.1. Combining the Existing Features

We combined some of the features in the training dataset to create new features [9]. For example:

- "area_total_calc"/"area_lot" gives the relative value of the total area and lot area for a property.

- "tax_building"/"area_lot" gives the building tax per unit area.

- "tax_property"/ "area_lot" gives the property tax per unit area.

3.4.2.2. Transformation of the Coordinates

We transferred the longitude/latitude coordinates of the properties from the Cartesian coordinates to the Polar coordinates (Figure 11). Additionally, we rotated the coordinate systems by 15, 30, 45, 60, 75 degrees, and added the new coordinates to the dataset. The notebook for performing these calculations is located under "./codes/ Location.ipynb".

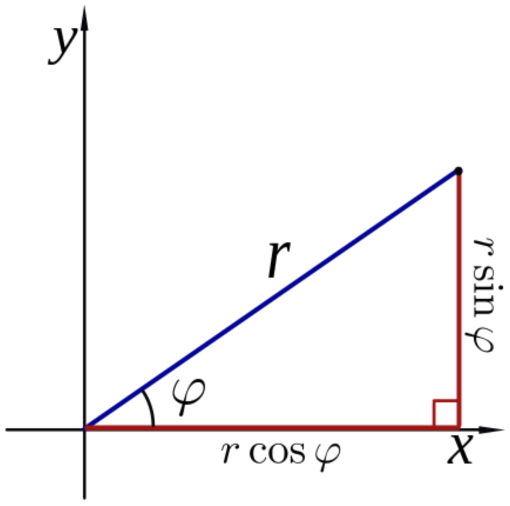


Figure 11. The latitude (X) and longitude (Y) coordinates of the properties were transformed to the polar coordinates. This figure shows the relationship between the polar and Cartesian coordinates [10]

3.4.2.3. Clustering

We hypothesized that Zestimate uses the sales values of the nearby properties to make predictions about the future sales prices of a certain property. However, when a property happens to be very different from its nearest neighbors, this method would likely fail. As such, a large logerror may be expected. For example, if we consider the 5 properties highlighted in Figure 12, 4 of them are relatively new but one of them is very old. Accordingly, the tax value of that property is also lower than its neighbors. The logerror value shows that Zestimate has overestimated the sales prices of that property. Therefore, we applied several clustering methods to distinguish such anomalies.

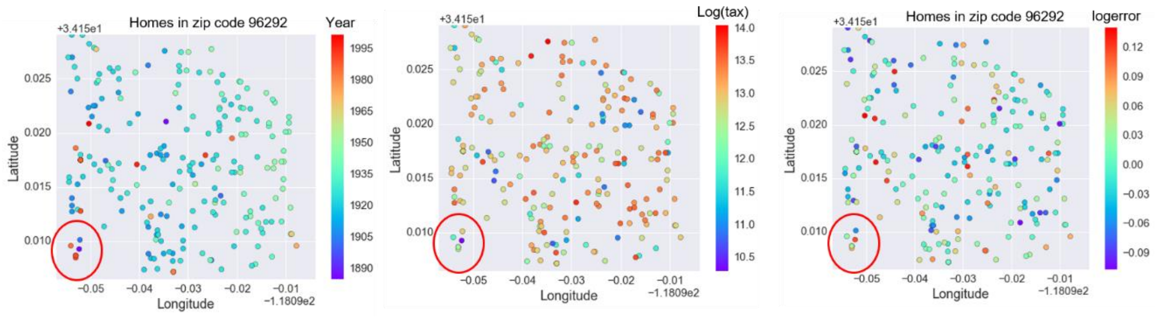


Figure 12. When a property is very different from its neighbors, Zestimate fails to accurately predict the sale price of that property. For example, the highlighted region in this figure contains 4 relatively new houses and 1 old house. Such inhomogeneity usually leads to a large logerror.

1. Clustering using existing features: we computed the average values of some important features including area_lot, tax_building, build_year within a zip code, city, etc. We then computed the deviation of a property from these average values, and added those values to the dataset. Furthermore, we computed the number of properties in each region.
2. Creation of new clusters: we created new clusters based on the latitude and longitude of the properties and similarly computed the deviation of a property from the average values. We tried to use the Silhouette score to select the optimum number of clusters. However, computing the Silhouette score for this dataset required large amount of memory. Accordingly, we relied on trial-and-error practices to arrive at the optimum number of clusters. Specifically, we tried the clusters of the size [80, 120, 290, 360, 470, 810], and eventually used 120, 290, and 360 (the code can be found under "./codes/Cluster.ipynb").
3. For each property, we computed the average values of some important features for k nearest neighbors. We optimized value of k was found to be 50 (the notebook for this calculations can be found under ("./codes/ k_nearest.ipynb")

4. Results

The optimum hyperparameters for our XGBoost model after an extensive process of feature engineering were found as followings:

- learning_rate: 0.01.
- max_depth: 9.
- num_round: 470.
- subsample: 1.
- min_child_weight: 70.
- colsample_bytree: 0.5.

The lower and upper boundaries for outlier removals were -0.252 and 0.264, respectively. These values implies that our model would make predictions that lie within a relatively narrow range (see Figure 14). It additionally shows that our solution is relatively robust, i.e., small changes in the input data will not significantly change the predictions. The effectiveness of our solution was further confirmed by testing its predictions for the unseen data on the competition's leaderboard.

The process of outlier removal and model optimization lowered the MAE from 0.0645278 to 0.0644122 (~0.2% improvement). Moreover, the extensive process of feature engineering further reduced the score to 0.0641420 (~0.5% improvement). These improvements are quite significant in Kaggle competitions because the scores of the top competing teams are extremely close to one another. This score is 3.5% better than the benchmark model that was obtained by using the average values (the score of the benchmark model was 0.0663010). The notebook "feature_engineering_2016.ipynb" performs all these steps and writes the augmented datasets into a binary file ("store_2016.h5"). The script "xgboost.ipynb" in the "codes" directory can then be used to fit a model to the data, calculate the score, and prepare a file for submission to Kaggle's website.

5. 2017 Data

The 2017 data were released on October 2, 2017. This data included transactions from January through mid-September, 2016. The teams were also provided with properties_2017 dataset. The 2017 dataset has the same parcel_id as in the properties_2016 file but the tax assessment values are updated based on the data that were collected in 2017 [11]. The features that we engineered for the 2016 data can seamlessly be applied to the 2017 data without any needs for further adjustments.

6. Conclusion

6.1. Free-Form Visualization

Figure 14 compares the distribution of logerrors obtained from our model and the training data. The plot shows that the predictions tends to be much narrower compared to the training data. This makes perfect sense because (i) we have dropped the outliers before fitting the models, and (ii) The model makes prediction by computing the average logerror over all values within a leaf. Although the values within the leaves have not been randomly selected, the distribution of the predictions resembles a sampling distribution. According to the Central Limit Theorem [12], the sampling distribution is normal with a standard deviation of $\sigma/\sqrt{n}$, where σ is the standard deviation of the population, and n is the number of samples.

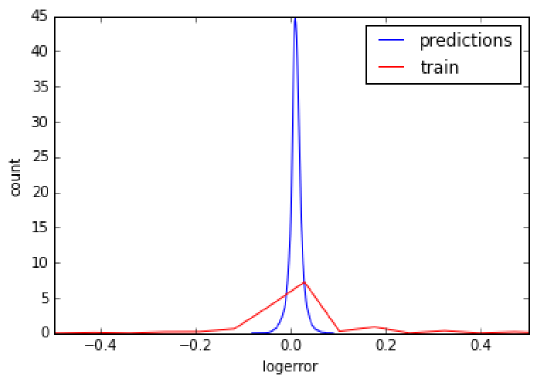


Figure 14. The distribution of logerror obtained from the predictions of our model is narrower compared to the distribution of the training dataset.

6.2. Reflection

Zillow's home value prediction competition with nearly 4,000 participating teams is among the most popular Kaggle competitions. In this project, we followed an extensive and iterative process to engineer new features, detect outliers, fit, and optimize our models. Figure 13 shows the workflow that was followed in this project. First, we examined the data and performed an extensive Exploratory Data Analysis (EDA) to realize the distribution of logerror, compute the correlation between features, and quantify the amount of missing values for each feature. Next, we imputed the missing values using different strategies. We also detected the outliers and properly excluded them from the training process. Last, we started our process of feature engineering and refined our model to arrive at the final model. Using very careful feature engineering and several weeks of continuous developments, we were able to achieve a competitive score on the public leaderboard.

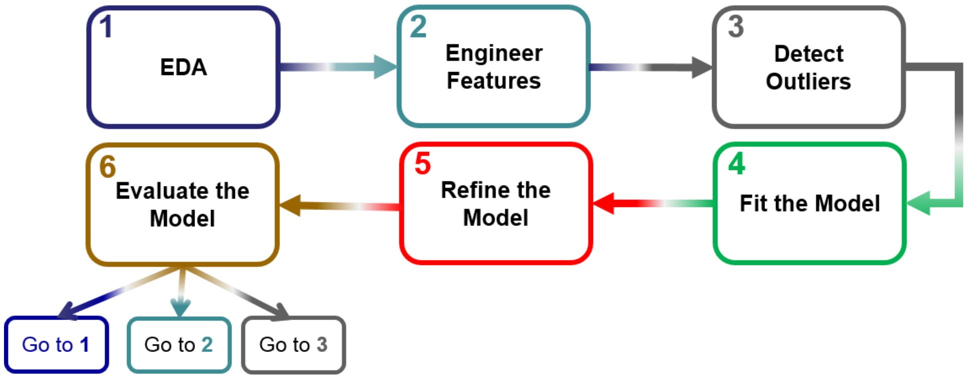


Figure 13. An iterative process was used in this project to engineer new features, detect outliers, fit, and optimize the models.

We realize that the process of creating a competitive model is definitely not straightforward. During this competition, we tested several ideas that did not work. For example, we divided the areas into square grids of various sizes, and computed the average values of important features within those grids. This strategy, however, did not improve the evaluation score.

Because our dataset is relatively large, some of the calculations required a lot of memory and highly specialized machines. Such machines are not readily accessible. As such, one has to use his/her intuitions and sometime compromise the accuracy of the models to obtain acceptable results within a reasonable time. Availability of powerful machines in the future will allow such demanding calculations.

6.3. Improvements

Although we have engineered numerous features in this project, there is always room for improvements. For example, once could perform Principal Component Analysis (PCA) [13] or

Independent Component Analysis (ICA) [14] to combine and transform some of the numerical variables into a low-dimensional space.

Because a single property may be sold several times, we predict that the Zestimate's logerror will likely show a systematically shift for the repeated transactions. Therefore, one could develop a procedure to handle these repeated transactions.

References

1. https://www.kaggle.com/c/zillow-prize-1

2. https://www.washingtonpost.com/realestate/zillow-faces-lawsuit-over-zestimate-tool-that-calculates-a-houses-worth/2017/05/09/b22d0318-3410-11e7-b4ee-434b6d506b37_story.html?utm_term=.17a115dcfad1

3. http://scikit-learn.org/stable/modules/model_evaluation.html#mean-absolute-error

4. https://www.kaggle.com/philippsp/exploratory-analysis-zillow

5. https://brage.bibsys.no/xmlui/handle/11250/2433761

6. https://github.com/dmlc/xgboost

7. https://en.wikipedia.org/wiki/Interquartile_range

8. https://en.wikipedia.org/wiki/Feature_engineering

9. https://www.kaggle.com/c/zillow-prize-1/discussion/39578

10. https://en.wikipedia.org/wiki/Polar_coordinate_system

11. https://www.kaggle.com/c/zillow-prize-1/discussion/40408

12. https://en.wikipedia.org/wiki/Central_limit_theorem

13. https://en.wikipedia.org/wiki/Principal_component_analysis

14. https://en.wikipedia.org/wiki/Independent_component_analysis

Appendix A

A.1. A sample of the "properties_2016" dataset

id_parcel	type_aircon	type_architectural	area_basement	num_bathroom	num_bedroom	type_framing	type_quality	num_bathroom_calc	type_deck
10754147	NA	NA	NA	0.0	0	NA	NA	NA	NA
10759547	NA	NA	NA	0.0	0	NA	NA	NA	NA
10843547	NA	NA	NA	0.0	0	NA	NA	NA	NA
10859147	NA	NA	NA	0.0	0	3	7	NA	NA
10879947	NA	NA	NA	0.0	0	4	NA	NA	NA
10898347	NA	NA	NA	0.0	0	4	7	NA	NA

area_firstfloor_finished	area_total_calc	area_live_finished	area_liveperi_finished	area_total_finished	area_unknown	area_base	fips	num_fireplace	num_bath
NA	NA	NA	NA	NA	NA	NA	6037	NA	NA
NA	NA	NA	NA	NA	NA	NA	6037	NA	NA
NA	73026	NA	NA	73026	NA	NA	6037	NA	NA
NA	5068	NA	NA	5068	NA	NA	6037	NA	NA
NA	1776	NA	NA	1776	NA	NA	6037	NA	NA
NA	2400	NA	NA	2400	NA	NA	6037	NA	NA

num_garage	area_garage	flag_tub	type_heating	latitude	longitude	area_lot	num_pool	area_pool	pooltypeid10
NA	NA	0	NA	34144442	-118654084	85768	NA	NA	NA
NA	NA	0	NA	34140430	-118625364	4083	NA	NA	NA
NA	NA	0	NA	33989359	-118394633	63085	NA	NA	NA
NA	NA	0	NA	34148863	-118437206	7521	NA	NA	NA
NA	NA	0	NA	34194168	-118385816	8512	NA	NA	NA
NA	NA	0	NA	34171873	-118380906	2500	NA	NA	NA

pooltypeid2	pooltypeid7	zoning_landuse_county	type_zoning_landuse	zoning_property	rawcensustractandblock	region_city	region_county	region_neighbor	region_zip
NA	NA	010D	269		60378002	37688	3101	NA	96337
NA	NA	0109	261	LCA11*	60378001	37688	3101	NA	96337
NA	NA	1200	47	LAC2	60377030	51617	3101	NA	96095
NA	NA	1200	47	LAC2	60371412	12447	3101	27080	96424
NA	NA	1210	31	LAM1	60371232	12447	3101	46795	96450
NA	NA	1210	31	LAC4	60371252	12447	3101	46795	96446

num_room	type_story	num_75_bath	type_material	num_unit	area_patio	area_shed	build_year	num_story	flag_fireplace
0	NA	NA	NA	NA	NA	NA	NA	NA	0
0	NA	NA	NA	NA	NA	NA	NA	NA	0
0	NA	NA	NA	2	NA	NA	NA	NA	0
0	NA	NA	NA	NA	NA	NA	1948	1	0
0	NA	NA	NA	1	NA	NA	1947	NA	0
0	NA	NA	NA	NA	NA	NA	1943	1	0

tax_building	tax_total	tax_year	tax_land	tax_property	tax_delinquency	tax_delinquency_year	censustractandblock
NA	9	2015	9	NA	0	NA	NA
NA	27516	2015	27516	NA	0	NA	NA
650756	1413387	2015	762631	20800.37	0	NA	NA
571346	1156834	2015	585488	14557.57	0	NA	NA
193796	433491	2015	239695	5725.17	0	NA	NA
176383	283315	2015	106932	3661.28	0	NA	NA

A.2. A sample of the “train_2016” dataset

id_parcel	logerror	date
11016594	0.0276	2016-01-01
14366692	-0.1684	2016-01-01
12098116	-0.0040	2016-01-01
12643413	0.0218	2016-01-02
14432541	-0.0050	2016-01-02
11509835	-0.2705	2016-01-02