Travis Flynn
October 18, 2017

Machine Learning Engineer Nanodegree
Capstone Project:

Zillow's Home Value Prediction (Zestimate) – Kaggle Competition

Project Overview:

Zillow is a real estate database website that utilizes statistical and machine learning models to provide estimates, which they term Zestimates, of current and future home market values for 110 million homes in the United States[1]. Homes are significant investments; having the most accurate information about their current and potential future value is of great use to buyers, sellers, investors, and real estate agents.

Since its launch in 2006, the team at Zillow has managed to decrease the median margin of error of their Zestimates from 14% to around 5% currently[2]. This has enabled Zillow to become a trusted source of information for 160 million monthly visitors across all of its brand websites[3].

The goal of this project is to participate in a competition sponsored by Zillow and hosted on the data science platform Kaggle. The competition's focus is on creating machine learning models that will lower the Zestimate error rate, thereby providing Zillow's users with even more reliable and accurate information.

Problem Statement:

The competition is separated into two stages. In stage 1, the goal is to produce a model that can predict the log error of a Zestimate for the months of October, November, and December in both 2016 and 2017, for each of the approximately 3 million properties in the dataset.

The value of the error is determined by subtracting the actual sales price of a home from the Zestimate. This is a supervised learning task in which the log error of real Zestimates has been provided. By producing models that can predict error rates on future home sales, Zillow's team hopes to identify methods for improving upon the already well functioning Zestimate algorithm.

I chose to approach this task with a focus on tree based regression algorithms as they are easy to work with, relatively fast trainers, somewhat robust to the outliers and random noise present in the data due to their ensemble nature, and work well with minimal preprocessing. My primary implementation was based on the regressor versions of CatBoost, XGBoost, and scikit-learn's Random Forest.

Stage 2 of the competition involves the creation of a new estimating algorithm that predicts actual home prices and that can defeat a version of the Zestimate algorithm that has been trained on the same dataset. Stage 2 is only open to the top 100 submissions from stage 1, so this project will analyze stage 1 exclusively.

Dataset:

The dataset for this competition is provided by Zillow. It consists of a full list of real estate properties in three counties (Los Angeles, Orange and Ventura, California) with data from 2016 and 2017. Information like tax value, bedroom and bathroom counts, square footage of the property, etc. are included in this dataset.

The three counties chosen represent a large area with a massive number of samples: 2,985,217 to be exact. There are 58 columns of features that describe the homes. The features come from county assessor's data, and Zillow. Most of the features are numerical, but some are categorical, and a few contain a mix of text and numerical data. This sample won't generalize across the entire U.S. but does serve as a primer for localized prediction in one of the largest real estate markets in the U.S.

The training dataset includes only those homes that actually sold during 2016 or 2017. The training dataset consists of a parcel ID that uniquely identifies the home, the sales date for the property, and the log error of the Zestimate for that sale. In total, 90,275 samples are provided in the 2016 training set. All sales up to Oct 1st are included. A small sample of sales from Oct, Nov, and Dec are present. The public leaderboard for this competition tests the remaining data from those three months.

For 2017 there is no data provided after September 30, 2017. The private and final leaderboard updates for stage 1 are scored on the last three months of 2017, updated roughly two weeks after the end of each month so that real sales data can be collected and Zestimate errors can be calculated.

To keep this report brief, I'll mostly focus on the 2016 portion of the competition when reporting values.

Evaluation Metric:

The evaluation metric for this competition has been chosen by Zillow. Evaluation of submissions will be on the mean absolute error (MAE) between the predicted log error and the actual log error[4]:

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

This particular metric was selected to ensure that high valued homes won't bias valuation models. The formula for MAE is given below:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} | y_i - \hat{y}_i |$$

Where :

n = number of samples i = the ith sample y_i = actual value $\hat{y}_i$ = predicted value

Data Exploration:

The provided dataset is a tabular, comma separated values sheet with nearly 3 million samples that have 58 features to describe them. It is actual real estate information from 3 counties in California. Zillow's data is obtained from publicly available sources, and since different cities and counties track statistics in different ways, it isn't surprising to find that this dataset has a number of features that are missing information. A few features are missing data for nearly every sample.

The features and what they represent aren't always immediately obvious from their names, but Zillow does provide a data dictionary that explains them. Here is a brief description of some of the features:

Feature	Description
airconditioningtypeid	Type of cooling system present in the home (if any)
architecturalstyletypeid	Architectural style of the home (i.e. ranch, colonial, split-level, etc…)
basementsqft	Finished living area below or partially below ground level
bathroomcnt	Number of bathrooms in home including fractional bathrooms
bedroomcnt	Number of bedrooms in home

buildingqualitytypeid	Assessment of condition of building from best (lowest) to worst (highest)
buildingclasstypeid	The building framing type (steel frame, wood frame, concrete/brick)
calculatedbathnbr	Number of bathrooms in home including fractional bathroom
decktypeid	Type of deck (if any) present on parcel
threequarterbathnbr	Number of 3/4 bathrooms in house (shower + sink + toilet)

The data dictionary also has breakdowns of what values in these fields represent. For instance, the feature airconditioningtypeid, listed above, can have numerical values like 1 or 3, representing central and evaporative cooling systems respectively.

Features like these present a challenge, not only because they have missing values, but also because a machine learning algorithm will assume the numerical value has significance. For example, a 5 can be mistaken as greater than 1, simply because of numeric difference, when no such difference is actually implied by the numbers for this feature. In fact, a 5 on airconditioningtypeid means there is no air conditioner at all.

Preprocessing for this dataset will have to include alterations to the features like replacing missing values (where applicable) and changing the way some of the features are represented via encoding strategies.

Based on the statistics for the log errors in the training datasets, outliers (some very extreme) are present. While there aren't many outliers in the training set (Fig.1, Fig. 2), because of their difference in magnitude, they represent an abnormally large amount of the MAE generated for the Zestimate. The mean log error is around 0.054 for non outlier samples (Fig. 2). The outlier samples have a mean log error that is around 12 – 20 times larger for any given month (Fig. 3).

Most Kaggle users that posted discussions or kernels for this competition were removing the outliers since doing so improved their leaderboard scores (indicating that the majority of the testing samples were typical homes that fall in the average range), but outlier removal results in the loss of real sales data. Since the goal of stage 1 is to model Zestimate error, I didn't want to remove the largest source of it. I decided to try to find a method to keep the outliers without letting their magnitude throw off the values of the typical homes.

Exploratory Visualization:

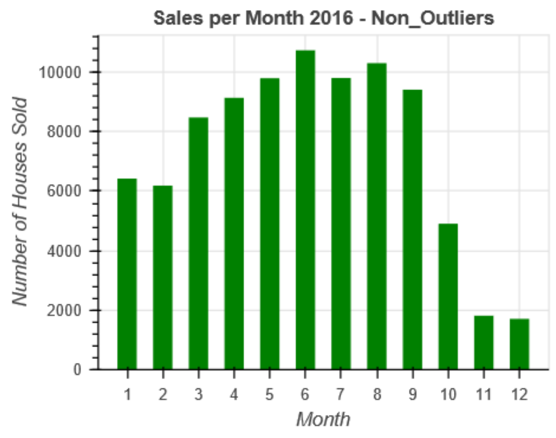


Figure 1. Sales per month in 2016 for non-outliers

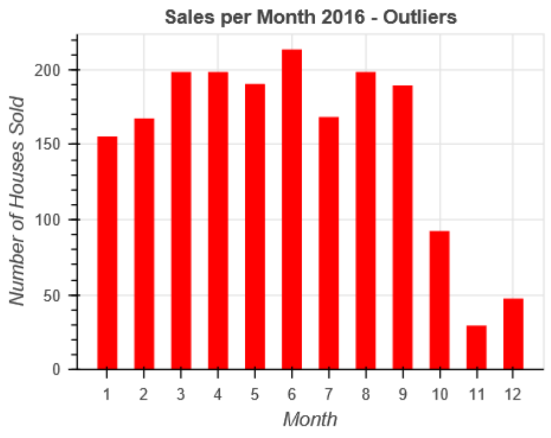


Figure 2. Sales per month in 2016 for outliers

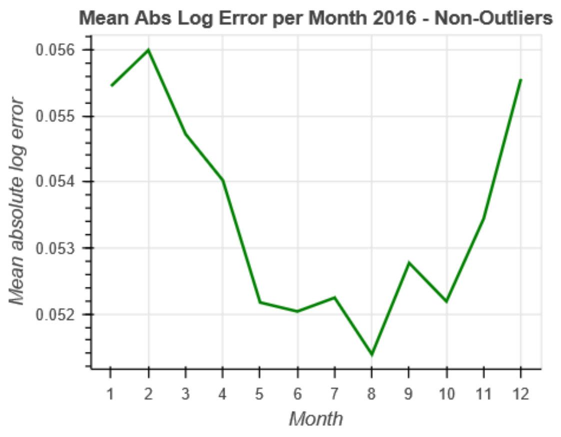


Figure 2. Mean of the absolute value of the log error for non-outlier samples in 2016.

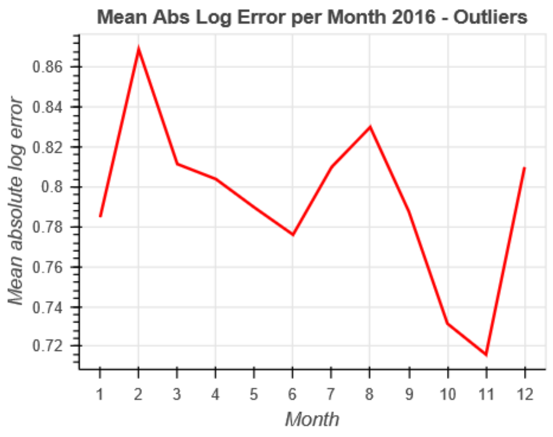


Figure 3. Mean of the absolute value of the log error outlier samples in 2016.

Cutoff points of -0.4 and 0.4 for the log errors were used to determine whether a sample was an outlier or not. This was more of a rough estimate than a traditional statistical definition of an outlier; utilizing this range produced better performing models. Samples outside this range were labeled outliers.

Figures 2 and 3 are produced on the absolute value of the log error, otherwise the graphs would misleadingly indicate that the log error was averaged at near zero. In all four figures, Oct, Nov, and Dec should be analyzed with some caution; as mentioned earlier, only a portion of sales data from those months was distributed with the dataset, the rest is part of the final testing set. Depending on how that data was selected, this could throw off the values for those three months significantly.

With those limitations kept in mind, the above figures still show an interesting trend, in that the error is smaller during the spring and summer months, when more homes are being sold. I felt that this was relevant, since the competition is asking for predictions during the colder months of Oct – Dec, when errors are likely to be slightly higher.

The difference in scale is also important to recognize. The indication is that the outliers represent a very minor portion of sales per month (which is why most Kaggle users were simply discarding them), but, they account for a large portion of the average error that the Zestimate produces due to the magnitude of the error. Unfortunately, a statistical analysis indicated that there were not large differences between the outliers and non-outliers in terms of features. Even that comparison was difficult, because the average values for roughly 1,844 outliers compared to 88,431 normal samples, presents significant potential for bias.

Despite this, I still felt it was necessary to see if I could find some way of modeling what causes a particular property to fall into the outlier range. I know that if that were a simple task, the team at Zillow wouldn't need to host a competition like this, since they'd already have corrected the greatest source of error already, but I wanted to give it a try nonetheless.

Algorithms and Techniques:

I began by creating a classification column in my training data frame that labeled outlier rows with a 1 and everything else with a zero. I then ran classifiers like logistic regression and classifier versions of CatBoost, just to see if there was a means of clearly deciding what an outlier was. If successful, I hoped to iterate through the entire properties list, finding those that were likely to be outliers, and score them with a separate regressor that was trained specifically for the outliers.

After trying multiple classifiers and feature sets, I achieved 97.7% predictive accuracy. Unfortunately, if I had simply labeled everything as a non-outlier, 97.96% accuracy was achievable. With predictions being made on all 3 million samples, the risk of incorrectly labeling non-outliers as outliers, just to find a few of the real outliers, was not justifiable.

When the 2017 datasets were released, I found that a small number of homes were sold during both years. For the most part the log error was similar. For a few homes, the 2016 estimate was nearly perfect, but the 2017 log error was vastly different. In one case, a home exhibited a 1,000 times difference in log error despite similar tax valuation and no other changes. Another home was a massive outlier in 2016, but nearly perfectly estimated in 2017, again with no apparent differences in the property itself. For these reasons, I was forced to accept that outliers could be entirely random. I abandoned my attempts to separate the samples due to time constraints and simply proceeded with the regression analysis.

My primary choice of regression algorithms were decision tree based ensemble models. I needed algorithms that would work with high dimensional data, were robust to the outliers that I fully intended to include, and could handle a mix of numerical and categorical data. Also, I had read that XGBoost and Catboost worked well “out of the box”, even with minimal preprocessing, but I had never used them before, so I wanted to see how well they would perform. They are both boosted tree algorithms with the option to create an ensemble simply by altering a parameter that increases the number of iterations.

A decision tree works by creating a set of rules that map feature input values to a targeted output value. For a highly simplified example, a decision rule for this dataset might ask if a property has a square footage feature greater than or equal to 2,000 sq ft. If yes, predict a log error of 0.005. If no, predict a log error of 0.003.

Features and values are chained together into a tree-like graph structure consisting of this pattern of if-then-else conditions until a final prediction is reached. The final prediction is essentially a path of inference based on the original features and corresponding target value for the samples in the training dataset.

A single decision tree has a tendency to overfit data by making an overly complex model. A random forest attempts to correct for this by creating an ensemble of decision trees that are constructed using random samples of the training data. Then, the predicted values of the individual trees are averaged for a final predicted value.

Gradient boosted tree algorithms work in a similar fashion to a random forest, but they rely on an ensemble of simplified models, termed “weak learners”, rather than fully grown decision trees. The learners are built in stages, with the goal of optimizing a loss function at each stage along a negative gradient[5]. These algorithms feature a learning rate that can control how quickly they converge toward an optimal value and regularization parameters that can help balance model predictive accuracy against model simplicity.

I hoped to try a neural network as well, but several Kaggle users had indicated that NN models were performing suboptimally for this particular dataset. Given the training time and computational power needed, I decided against trying one.

Benchmark:

Since I had decided on using ensemble tree algorithms, the most appropriate benchmark was a single decision tree. I implemented scikit-learn’s DecisionTreeRegressor class and created a custom MAE scorer to use as my benchmark. The benchmark model, trained on my final cleaned version of the dataset, scored as follows:

Mean Cross Validation MAE: 0.06040

Test MAE: 0.05883

Data Preprocessing:

CatBoost and XGBoost did turn out to have strong performance with little preprocessing, but I chose to go through multiple preprocessing stages anyway, just to see if I could improve performance. Ultimately, I treated the preprocessing phase as though it were part of my model selection, testing different methods to see what effect they had on cross validation scores for the benchmark model and a random forest model.

I initially dropped multiple columns from my dataset simply because Zillow calculates many of them into an aggregated value. For instance, the county assessor's data includes values for the number of bathrooms, three-quarter bathrooms, half bathrooms etc., where as the Zillow column calculatedbthnbr aggregates them. Since the Zestimate algorithm uses the aggregated columns, I only kept those.

Many of the remaining features contained missing values. I chose to eliminate any feature that was missing more than 50% of the data. From there, I experimented with dropping NaN values or filling them with the mean, median, or mode values of each feature in question. My best results were achieved by filling numerical values with the median and categorical values with the mode.

Numerical values were scaled using scikit-learn's StandardScaler and categorical values were label encoded with the LabelEncoder class.

I tried submissions to the competition with and without outliers. Submissions without them were the highest scoring until I decided to try compressing the outlier's values. I initially divided log error of any sample outside the -04. - 0.4 range by 10, but after iterative steps, I found that compressing them to half of their original value gave the best results. This allowed me to keep the outlier's information without having their inclusion significantly alter the results for normal samples. My highest scoring submission to the competition was achieved with this method.

Given the high dimensionality of the data, one hot encoding of categorical values proved to be an issue. I tried a mix of ohe and PCA decomposition, and selecting best features based on information from the tree models. I ultimately settled on using the best features from a Random Forest model. This gave consistent performance and left the dataset with mostly numerical features.

The only feature engineering I did was to separate the sales date into a transaction month column.

Implementation:

I began by splitting the data into training and testing splits. From there, I fit the imputer, scaler, and encoder objects mentioned above on the training data, and used those to transform both training and testing sets.

The transformed training set was then fed into scikit-learn's cross_validation_score module with 5 folds, the custom MAE scorer created for the benchmark, and one implementation for each of the following models: RandomForestRegressor, XGBoost, CatBoost, and LinearRegression. The linear regressor was really only used as a second benchmark since it was easy to implement and very fast to train.

I tried to keep initial parameters as close to one another as possible for the tree models; 500 iterations, and similar regulation and learning rates where applicable. I evaluated the raw cross validation results, the averaged cross validation results, and the score predicted on the testing set to decide on a final model to tune.

CatBoost came out ahead of the other models by a slight margin. The linear regressor was significantly faster than all three tree models, but offered the lowest performance, as I expected.

Initial implementation of these algorithms was straightforward. All of them worked without any issues other than long run times for the cross validation portion. Tuning the model proved to be a far more troublesome step.

Refinement:

While I did try minor tuning on all three tree models, I spent the most time trying to improve the CatBoost model. My initial choice was to implement the GridSearchCV module from scikit-learn and feed in a set of parameter variations to test. The tuned parameters were selected from the CatBoost website which lists parameters that typically help performance and those that rarely make a difference[6]. I chose to ignore the parameters that they listed as rarely useful.

I created the GridSearchCV object and left it for a few hours to do its work. When I came back to check on it, it had crashed the kernel (I was running this in the Spyder environment provided with Anaconda), essentially saying the system had run out of memory. I assumed it was a fluke, since I had set up the GridSearchCV class the same way I have in other projects, so I tried again the next day, with the same result. I tried a third time, this time keeping an eye on ram usage. Within seconds of starting the search, ram usage was at 100%.

After some digging on Stack Overflow, I found that this is a common issue with GridSearchCV and tree based models if you are using the parameter n_jobs = -1, which distributes the search across all processing cores. The issues appears to be that it creates a huge number of copies of the trees in system memory to distribute, almost guaranteeing that ram will be completely consumed. Since I'm running my analysis on a relatively inexpensive laptop, my memory was completely consumed in seconds. The only workaround was to use only a single cpu core, which then imposed crippling time restrictions.

The 2017 training set was released on Oct 2nd, with the final submission deadline on Oct 16th. I had already used much of that time analyzing the new data, and then several days trying to figure out why the GridSearch wasn't working. I no longer had time to try an exhaustive search on one cpu core.

So, my parameter tuning came down to more of a randomized search on just a few parameter / value combinations. Here is an example of some of the hyperparameters that I tried, with associated scores:

iterations	learning_rate	depth	l2_leaf_reg	CV MAE	Test MAE
500	0.03	4	4	0.059824	0.058533
1000	0.03	4	3	0.059823	0.059016
1200	0.03	4	3	0.059860	0.059053
1000	0.06	4	3	0.060378	0.061200
1000	0.005	4	3	0.059707	0.058291
1000	0.005	4	1	0.059740	0.058406
1000	0.005	4	10	0.059740	0.058173
1000	0.005	4	15	0.059711	0.058179
1000	0.005	5	15	0.059692	0.058163
1000	0.005	6	15	0.059710	0.058233

The depth 5 model had the highest score on the CV and test sets, but produced slightly lower competition scores than my depth 4 model with all other values held equal.

I was somewhat surprised to find that the “works out of the box” mentality did hold up, more or less. Though I did manage to select parameters that increased my cv, test, and competition results, they weren’t vastly different than a simple default configuration with the same number of iterations. Both CatBoost and XGBoost were quite capable with very little tuning at all. The initial and final model parameters that I used are as follows:

Initial	**Final**
CatBoost(iterations=500,	CatBoost(iterations=1000,
learning_rate=0.03,	learning_rate=0.005,
depth=4,	depth=4,
l2_leaf_reg=3,	l2_leaf_reg=15,
loss_function='MAE',	loss_function='MAE',
eval_metric='MAE',	eval_metric='MAE',
random_seed=10)	random_seed=10)

Model Evaluation, Validation, and Justification:

The final model scored the following on the CV and testing sets:

Mean Cross Validation MAE: 0.05971

Test MAE: 0.05817

This is an improvement over the benchmark model of 0.00069 and 0.00066 respectively. These seem very minor, but this competition is impacted by even the smallest of gains.

My train and test splits were created before imputation or scaling to ensure that the test split would be as close an approximation of a real test set as possible. My final model performed well on this split and the 5 CV folds that I ran on the training data. It also produced my highest score on the competition’s public leaderboard. Based on CV testing for the 2017 data, I think that it will produce similar results on the private leaderboard as it gets updated over the next three months.

I tried to stick with a simpler model with a tree depth of 4 and default leaf parameters. The learning rate is small, but should help it avoid overshooting an optimal convergence point. Small changes in the training data did not produce significantly different results. I even tried plugging in the 2017 training data into my 2016 trainer and then predicting the 2016 test results just to see what would happen. It still performed well, not quite as well as the proper 2016 training data, but close.

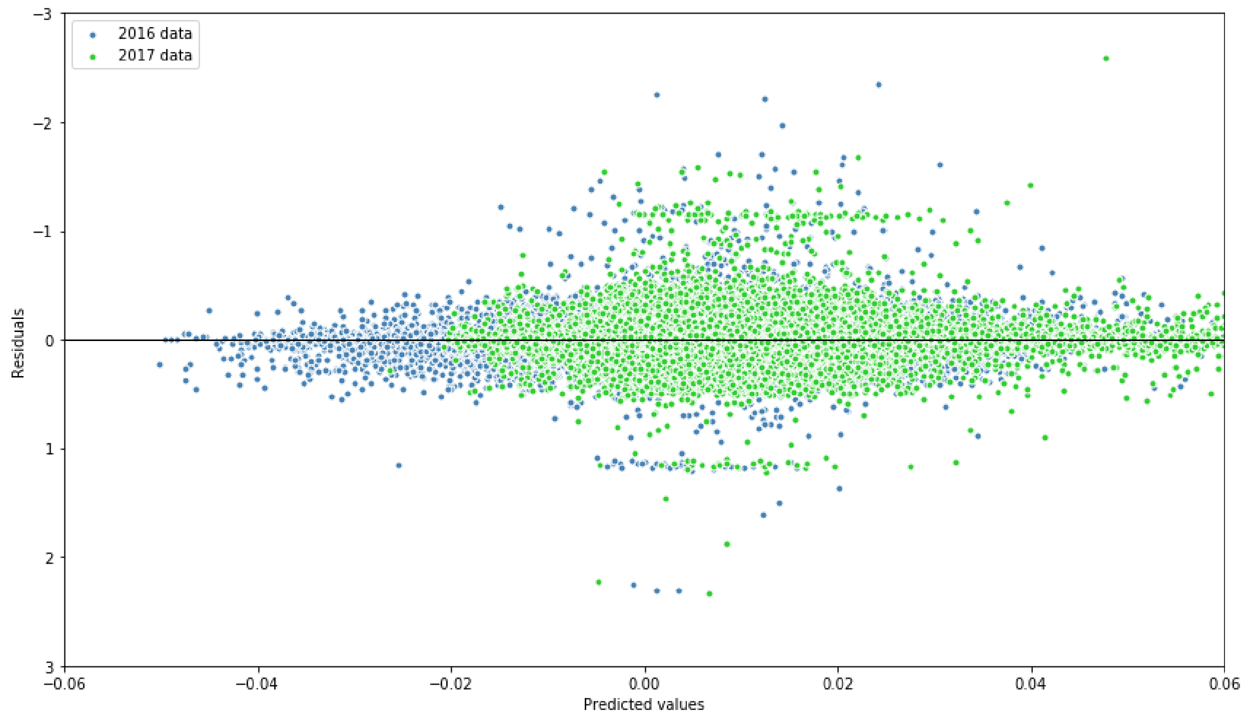


Figure 5. Residuals for predicted values of trained model for 2016 and 2017.

The training set for 2017 was highly similar to that of 2016, and given that the plot of the residuals for 2016 and 2017 indicate that my final trained model produces similar patterns for both years, I expect the performance of the model in the private leaderboard for 2017 to closely reflect that of 2016's public leader board.

At the time of writing, the top model on the public leaderboard has a MAE of 0.0631885. My final model produced a MAE 0.0645506, a difference of 0.0013621.

Free-Form Visualization:

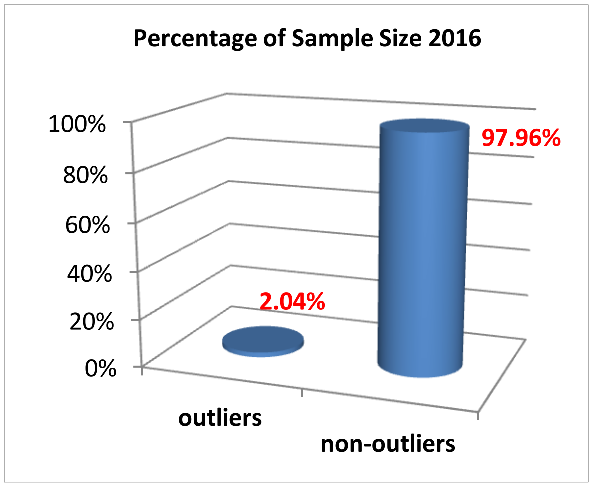


Figure 6. Sample size by outlier designation 2016

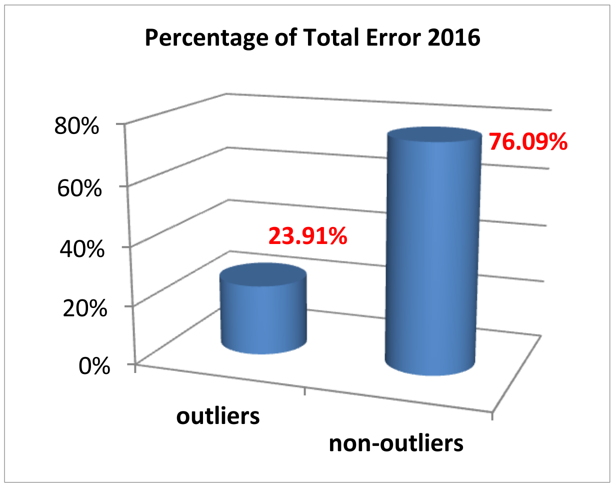
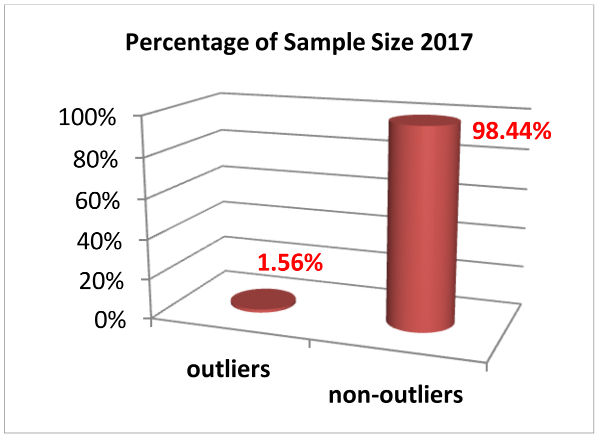


Figure 7. Total error by outlier designation 2016



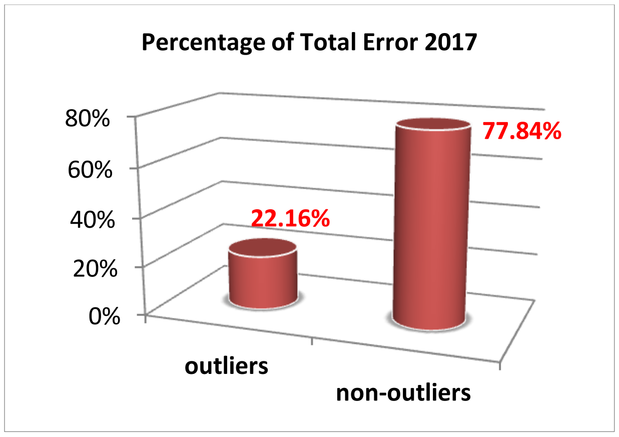


Figure 8. Sample size by outlier designation 2017

Figure 9. Total error by outlier designation 2017

The above figures illustrate the most challenging aspect of this entire competition; a very small subset of properties represent more than 20% of the total error generated by Zillow's Zestimate algorithm year to year. With the given data, there was no reliable way to discover why these particular properties are outliers, or predict whether they will be again, when they next come up for sale.

I was able to compress their effects and utilize their training information to increase my predictive accuracy, but I still can't offer more than a guess as to why they exist. It could perhaps come down to something as simple as a motivated owner selling a house for much less than it is worth, or someone buying their dream house at a higher price than they should have. Unfortunately, neither event is one that I can model.

Reflection:

The following process was used followed to create this project.

1. Import Packages and Datasets
2. Data Exploration and Statistical Analysis
3. Data Preprocessing
4. Construct and Evaluate Benchmark Model
5. Feature Selection and Model Evaluation
6. Final Model Tuning
7. Final Model Training and Prediction
8. Competition Submission and Results Analysis

Many of the above steps were iterative, as I made small adjustments a number of times to see if better competition scores could be achieved.

Overall, I found this to be a very interesting project. Making predictions on 3 million samples across 6 time periods without knowing which of the samples will actually be tested made for a fun, and occasionally frustrating, challenge. The fact that it is real data, and not simulated data makes it all the better.

I had no illusions about winning the competition, I set out to produce a good model and try something new along the way. I tried CatBoost and XGBoost, and learned that they are very powerful tools. I used Bokeh to construct most of my graphs, rather than matplotlib, again, just to try something new. I learned that GridSearch, parallelism, and tree based algorithms don't always play well together. And ultimately, I think I produced a functional model.

Improvement:

I believe I was close to squeezing out as much performance as possible from the raw features that I was using. I think whoever wins this competition will have done superior feature engineering. For instance breaking down the sales date into day, week, and month columns rather than just a transaction month column, would probably have been beneficial.

Engineering new information from the location data, especially when combined with outside information sources would also be beneficial. For instance, is the property near schools, hospitals, transportation hubs, or popular venues? Outside information was prohibited for stage 1 of the competition, but I'm sure I could have done more with the location data than just use it in its raw form.

In the end though, it's the detection of the elusive outliers that would produce the greatest gains. They represent actual sales, but only around 1- 2% of them, yet they account for ~22% of the Zestimate error. A reliable method of predicting which properties are likely to be outliers for any given month would greatly increase the predictive accuracy of any of the models that I trained.

---

1. https://www.zillow.com/corp/About.htm

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

3. https://www.zillow.com/advertise/

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

5. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor

6. https://tech.yandex.com/catboost/doc/dg/concepts/parameter-tuning-docpage/