Real Estate Price Regression

Routhana Long

CPTS315 Course Project

Fall 2018

Dr. Kyle Doty

Table of Contents

Introduction

Zillow's Zestimate application, a real estate price estimation service, has been under increased scrutiny for inaccurately, underestimating real estate prices. This, in turn, led to home buyers using the service to negotiate for lower prices leaving home owners and sellers at a loss of selling power. These frustrations ultimately spiraled into formal complaints and legal disputes including class action lawsuits in recent years.

This project intended on examining the real estate prices in the King County area—the goal being to create a regressor to predict real estate prices sourced from a Kaggle data repository. In the process, it explored and engineered features that potentially impact housing prices and algorithms and ensemble methods that could improve regressor performance.

The biggest constraints in this project were related to the data. In comparison to the Zillow dataset, the King County dataset is notably shallower in terms of overall data samples and quality features encompassing subjects such as neighborhood quality that were otherwise available in the Zillow dataset.

Ultimately, feature engineering of distance and travel duration data immediately resulted in a median absolute error (MAE) improvement of $859.08. While exploration of the stacked ensembling method proved to be a costly and fruitless endeavor whose application is niche, parameter tuning of a lone XGBoost model improved the MAE by $1,689.97 all the while being able to provide a prediction in 0.04 seconds.

Data Mining Task

The data mining task was to engineer an accurate and reliable real estate price predictor based on a variety of features included in Kaggle's House Sales in King County, USA dataset, since there were issues obtaining permission to use Zillow's very own dataset available on Kaggle through a previous year's competition.

The models were trained on a variety of traditional and non-traditional real estate data. From the original data set, these features include rooms, bathrooms, square footage, build year, number of floors, etc. which are sourced through a comma-separated file.

The goals of this project were to engineer features that hopefully would improve model performance and to compare the performance of various models and/or methods including XGBoost and stacked ensembling. Specifically, the feature engineering goals were to answer the question, does location really matter? To answer this question, the creation of locality-based features was explored, and these new features were analyzed. These features included travel (road) distance from major city centers and airports as well as the time to travel to major city centers, as well as nearest geodesic distance to the i5 freeway.

Since Zillow's Zestimate service is a real-time application, the application should make real estate estimations in a timely and accurate manner.

Technical Approach

Feature Engineering

Upon exploratory data analysis of the 21,614 housing prices, it appeared that housing prices were lower when they were closer to freeways and airports and higher when they were closer to major city centers. Thus, features were engineered with duration and distance in mind. Travel duration and travel distance from a city center utilized Google's Maps API through its Python library to return the two-variable for each property to some point in the city. The cities were Seattle, Bellevue, and Redmond.

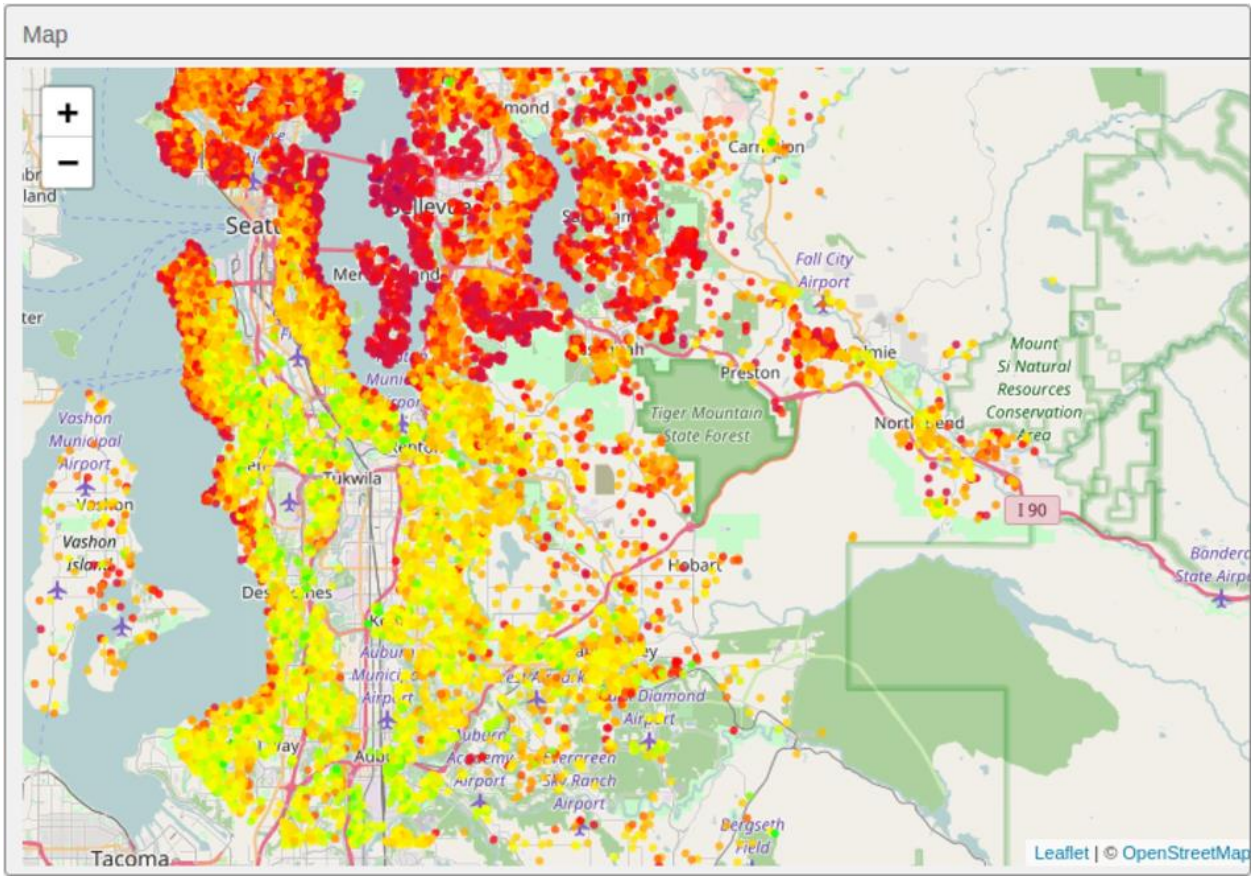

Figure 1. Map of real estate listings. Green indicates lowest prices, purple highest prices.

Normal distance, on the other hand, takes the geodesic distance from the latitude and longitude of a location, which were hand labeled on a very slow day by *driving* up the i5 freeway on Google Maps or by identifying certain locational landmarks. These locations included Seattle-Tacoma International Airport, Boeing Field, and Renton Airfield. These distances utilized the Vincenty method for geodesic calculations which was accessible through the GeoPy library and solves for distance iteratively on the surface of a sphere (Earth) [1].

[1] Bessel, F. W. (2012). The calculation of longitude and latitude from geodesic measurements∗. ArXiv, 1-11. doi:10.1002/asna.201011352

Cross-validation and Model Selection

Cross-validation is an algorithm that is related to the Central Limit Theorem. The algorithm takes a training set and an input of k amount of folds and proceeds to partition $\frac{k-1}{k}$ of the data to train the model and the other $\frac{1}{k}$ of the data for validation. This process continues by iterating through k possible folds of data until all possible fold train-validation combinations have created models. Ultimately, this results in k models trained, in which the mean of some performance metric such as r^2 or $RMSE$ is taken to compare against other or choose optimal models, parameters, etc.

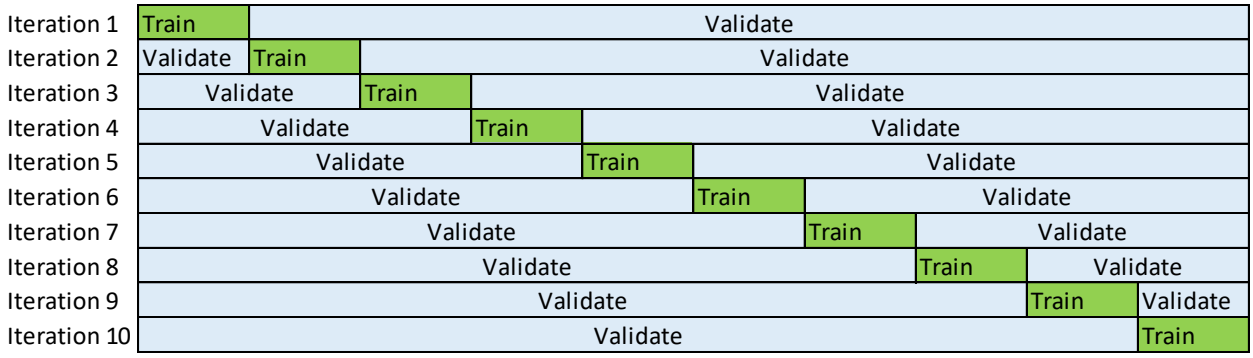

Figure 2. Train and validation of training set in 10-fold cross validation.

By using cross-validation, models are less prone to overfitting, or high variance, which is the case where a model fits the data too well and generalizes on unseen data poorly. This overfitting aversion is attributed to the process of iterating through folds and training on different partitions in an attempt to diversify the data that the models are trained upon, and thus the models themselves.

The training set comprised of all but five randomly selected data samples from the original data set that was excluded for the sole purpose of testing the final application. This training set was used in 10-fold cross validation which trains each model on 90% of the training set and validated against the remaining 10% across 10 iterations of cross-validation folds. These 10 validation scores would then be averaged for each model to identify optimal hyperparameters for each model as well as optimal models for specific algorithms themselves.

Support Vector Machines and Decision Trees

The models used include support vector machines (SVM) and a decision tree in regressor roles to use in and compare against the new techniques explored. SVM is an algorithm that uses a kernel to exploit mappings of features and spaces to try to find decision boundaries, and in the case of regression, it could make boundaries reminiscent of time-series bands (e.g. Bollinger bands) in a linear space. The SVM algorithm was tested against multiple degrees of polynomials as well as the penalty parameter C to attempt to find an optimal model. Likewise, the decision tree, which is reminiscent of other data structure tree algorithms but holding questions such as *is x greater than 50*, was optimized to maximum depth, minimum leaf samples, and minimum split samples.

Boosted Ensembles, Gradient Boosting, and XGBoost

Historically, boosting refers to pushing up or supporting an object. In the context of machine learning, boosting refers to the conversion of weak learners into strong learners. This can be done through a multitude of methods such as average, weighted, or ensembled voting.

Boosting does this by creating random subsets of samples from a training set (similar to bagging) and drawing a random subset of samples from a subset without replacement from the training set to train a weak learner. Then, it chooses a different subset from the training set in addition to some percentage of the samples from the previous training instance that did not meet some metric and trains a new learner. After that, it trains yet another learner, this time on some training set where the previous two learners disagree and from these different learners, it creates an aggregated learner by combining the strengths of all the learners involved[2].

Gradient boosting is an extension of the previously mentioned boosted ensemble model. It uses gradient descent, a loss optimization algorithm, with boosting to fit additive models increasing performance in both accuracy and in speed. XGBoost, which is the primary algorithm in this experiment, extends upon gradient boosting all the while using more regularized model formalization to control overfitting[3].

Stacked Ensembling through a Regressor

The models that were used are decision tree regressor, support vector machines (SVM), and XGBoost. In addition, combinations of all models were fed into a stacking regressor resulting in machine learning models for the decision tree regressor, SVM, XGBoost, and the stacked ensembles. For all models, parameters were tuned manually.

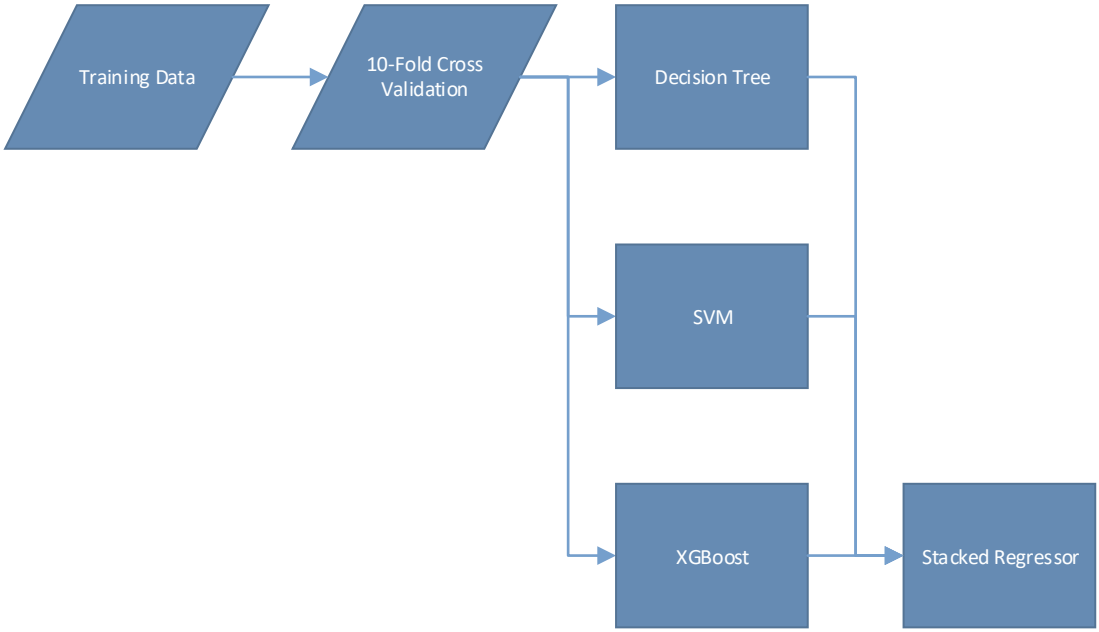

Figure 3. Stacked ensemble regressor (training).

[2] Ray, S. (2015, November 9). Quick Guide to Boosting Algorithms in Machine Learning. Retrieved December 6, 2018, from https://www.analyticsvidhya.com/blog/2015/11/quick-introduction-boosting-algorithms-machine-learning/

[3] Chen, T. (2015, September 3). Re: What is the difference between the R gbm (gradient boosting machine) and xgboost (extreme gradient boosting)? [Web log comment]. Retrieved December 6, 2018, from https://www.quora.com/What-is-the-difference-between-the-R-gbm-gradient-boosting-machine-and-xgboost-extreme-gradient-boosting

Evaluation Methodology

There are multiple levels of evaluation in this project and this comes down to model evaluation and application evaluation. The model needs to have sufficient pricing accuracy while the application should not result in the user waiting minutes for a price.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| = \frac{1}{n}\sum_{i=1}^{n}|e_i|$$

Being a regression problem, model evaluation is limited to regression metrics. These metrics include r^2 score, mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), and so on. For model selection and parameter tuning, MAE was used as the data is notably skewed and thus a metric that is least susceptible to outliers works best. MAE works best because of the median's way of somewhat counteracting the outliers and data skewness[4].

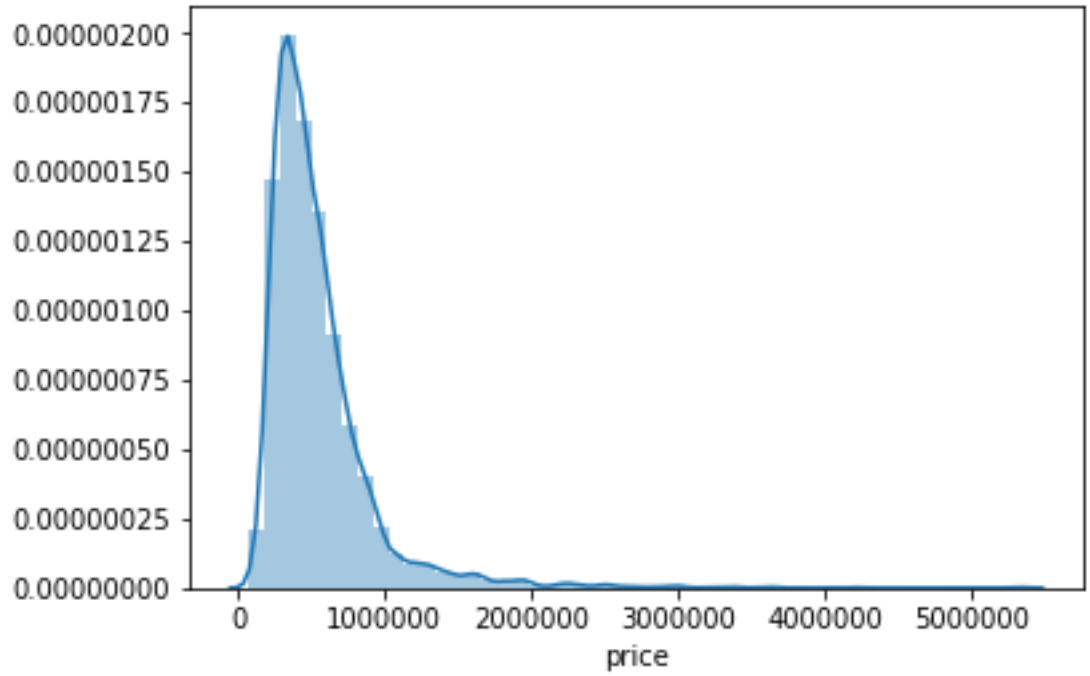

Figure 4. 2015 real estate price distribution. Skewed right hence the decision to use the MAE metric.

As for application evaluation, the slower the website, the more likely the website is to be abandoned by its user. Though we are not developing a website, we hope to develop an predictor that produces a prediction within 3 seconds, since there is a 40% chance of website abandonment if loading times are greater[5].

Results and Discussion

Preliminary Model and Feature Engineering

[4] Swalin, A. (2018, April 07). Choosing the Right Metric for Evaluating Machine Learning Models - Part 1. Retrieved December 6, 2018, from https://medium.com/usf-msds/choosing-the-right-metric-for-machine-learning-models-part-1-a99d7d7414e4

[5] Work, S. (2017, August 24). How Loading Time Affects Your Bottom Line. Retrieved December 6, 2018, from https://neilpatel.com/blog/loading-time/

A preliminary model was trained on XGBoost using the original dataset which resulted in an MAE score of $37,109.89. After hand labeling latitude and longitude data for the i5 freeway, these latitude and longitude values were used to compute the minimal geodesic distance to the i5 freeway. The process takes about 1 hour and 20 minutes to compute all distances and the result of this new feature resulted in a new MAE score of $36,678.03 an improvement of $431.86.

Centroids of Seattle, Bellevue, and Redmond were then identified to be (47.6475, -122.3497), (47.6101, -122.2015), and (47.6740, -122.1215), respectively, and using the paid Google Maps Python API, travel durations and distances were retrieved from every property. A preliminary XGBoost model on this new data including the data before improved MAE to $36,302.25 or an improvement of $807.64, while the addition of distances to Boeing Field, Seattle-Tacoma International Airport, and Renton Airfield improved MAE to $36,250.81, an $859.08 improvement. Feature importance also showed that the new features, though not as important as square footage features, provided a high number of splits, hence some added value to the data set (figure 5).

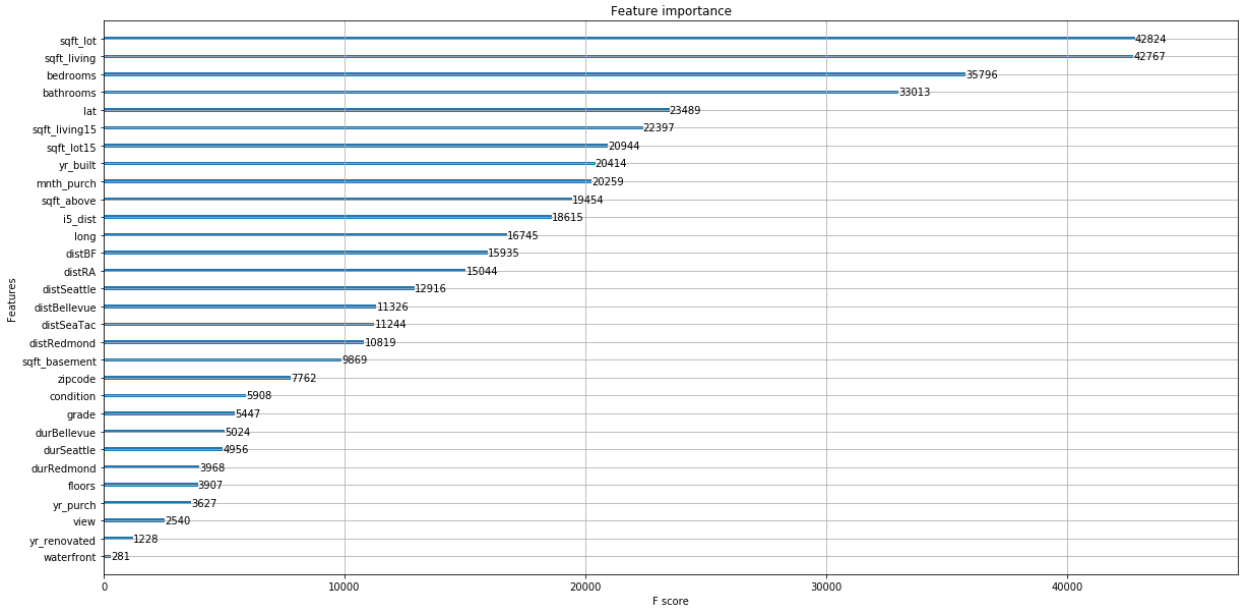

Figure 5. Feature importance. F-score is based on the number of splits by the XGBoost algorithm. The new features are great mid-level features.

Models and Parameter Tuning

Decision tree regressor parameter optimization entailed examination of three variables: maximum depth, minimum leaf samples, and minimum split samples. Ultimately, maximum depth and minimum leaf samples provided the most variations in performance. The best parameter values came out to be a maximum depth of 12, minimum leaf samples of 11, and minimum split samples of 2 resulting in a cross-validation mean MAE score $48,653.50 which is worse than the baseline XGBoost model but is not unexpected being a single tree.

Models based on support vector machine regressors proved to be less performant. They were both notably slow to train and had very high MAE values. The parameters tuned for SVM models were degree and penalty parameter C. Degrees were insignificant, while lower C values were generally better (figure 7). The best SVM model had a cross-validation mean MAE of $149,795.45, which is more than $100,000 higher than the best decision tree and more than $112,000 worse than the baseline XGBoost model.

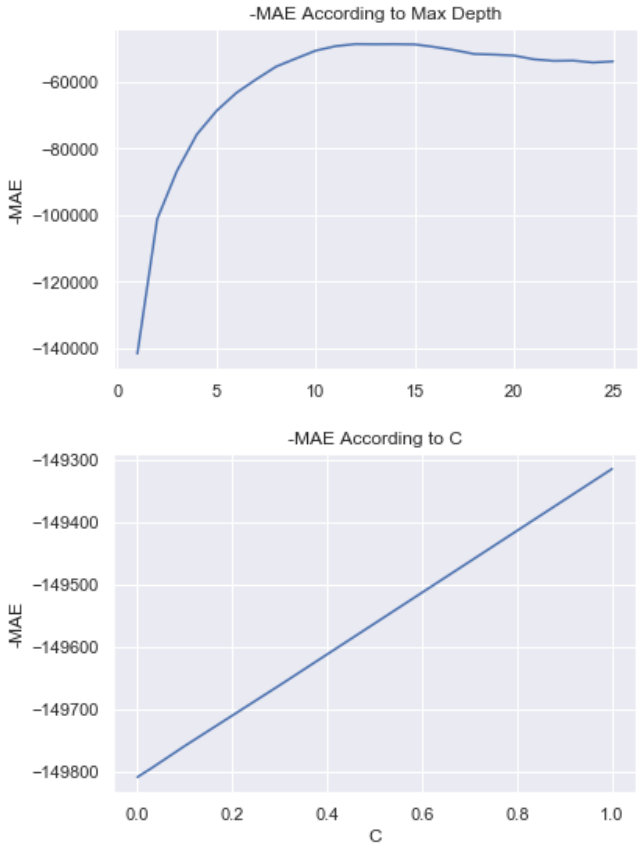

Figure 7. -MAE score vs C. Higher is better. Overall a worse model for the data inputs.

XGBoost models were the most responsive to parameter tuning but was unfortunately similarly slow to SVMs to train as the number of trees (estimators) grew. Thus, only three hyperparameters were tuned: the number of estimators, the minimum child weight, and the maximum depth—with the latter two being limited to cap the amount of time it would run. Ultimately, the most optimal parameters were n_estimators=2750, min_child_weight=1, and max_depth=9, resulting in a mean CV MAE of $35,419.92, a $1,689.97 improvement over the base XGBoost model.

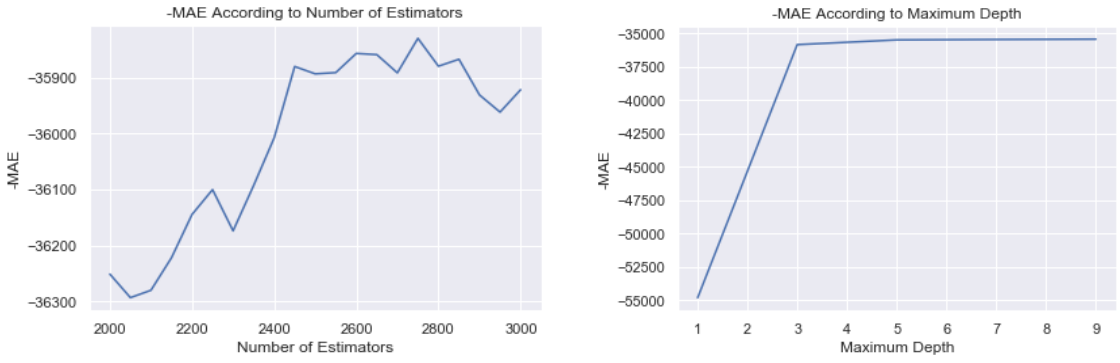

Figure 8. -MAE vs XGB Number of Estimators and Maximum Depth. Higher is better. Optimal estimators=2750. Optimal max depth=9.

The three models and their optimal parameters were finally used in a stacked ensemble, feeding combinations of the three models into a regressor. The results, as seen in figure 9, are ensembles which are noticeably worse than the XGBoost models. It should be noted, though, that some stacked ensemble combinations improved from the models that fed into the regressor. The stacked ensemble featuring SVM and decision tree improved to $48,651.87 from $48,653.50 for the decision tree and $149,795.45 for the SVM models, which is a miniscule but very intriguing result given the drastically bad performance of the SVM model alone.

Overall, stacked ensembling seemed to have a little bit of randomness to it. Thus, it is a technique that can bring incremental gains with a lot of added complexity, both in time (training) complexity and in architecture[6]. So, the best model for this specific data set given experimentation is the tuned XGBoost model.

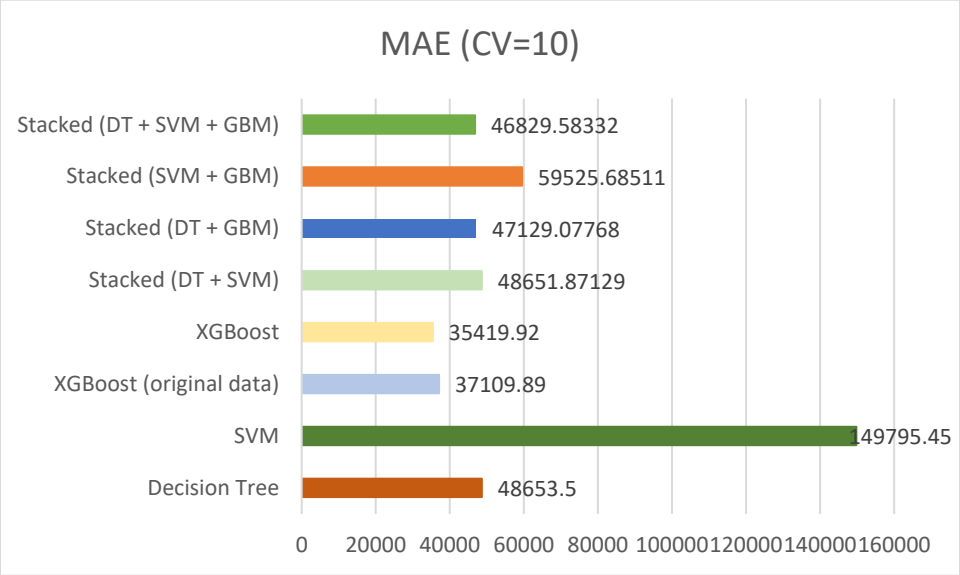

Figure 9. Mean median absolute values of all models across 10-fold cross validation.

The final application based on XGBoost results in price being predicted in 0.04 seconds (Intel i7 2600k processor, 16GB memory), with the predictions of 5 samples being in a range of $12,901.50 to $136,816.84 within their actual values (figure 10). This equates to an MAE of $32,643.94 for a **very small sample**.

[6] Gorman, B. (2016, December 27). A Kaggler's Guide to Model Stacking in Practice. Retrieved December 6, 2018, from http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/

	Actual	Predicted	Absolute Error	log(error)
1	$572,000.00	$435,183.16	$ 136,816.84	0.12
2	$475,000.00	$541,374.40	$ 66,374.38	0.06
3	$350,000.00	$369,459.88	$ 19,459.88	0.02
4	$362,500.00	$375,401.50	$ 12,901.50	0.02
5	$438,000.00	$470,643.94	$ 32,643.94	0.03
MAE	$			32,643.94
Prediction Time (s)				0.040624

Figure 10. Test results. Home owners 1 and 2 will not be happy customers.

Unfortunately, the results and metrics of Zillow's Kaggle competition are based on the difference in $\log error$ which is $\log Zestimate - \log SalePrice$. This implies that the $35,419.92 MAE at the median household value for the King County data set of $450,000 would be around 0.02 in $\log error$, which unfortunately cannot be further examined respective to Zillow values due to lack of consent. In addition, without the Zillow data set, including house values, it is difficult gauge where these features and models stand as a product to end users, but overall, the XGBoost model has proven to be very speedy and relatively accurate.

Lessons Learned

In hindsight, there were a mixture of plans that had to abandoned due to time constraints and complications. This comes from a lack of exposure and naivety to model training.

Parameter tuning via the grid search algorithm was one of the things that had to be left out as combinations of parameters on all models excluding vanilla decision trees took days to optimize (SVM was keyboard interrupted at 7 days). Thus, better understanding and exposure to the time complexities of each task could have aided to significantly better time management. Regardless, it was amazing how time could be optimized to fit around model training and optimization particularly with larger, computationally intensive models which I had previously only heard stories of (e.g. when working on deep nets). Thus, time management in a machine learning project was really emphasized by the end of this project.

Otherwise, there were a great number of topics learned through this project. New ideas such as gradient boosting and stacked ensembling were some of them. Other ideas included proper metrics selection and the generalization that all models do not perform similarly on the same data set, and that learning rate could eek out even more performance by more precisely converging. And finally, as evident by XGBoost, ensembling can generally have a major impact on model performance by exploitation of weak learners.

Beyond those possible improvements and the topics I learned lies the emphasis on the quality of data. The Zillow dataset is a much more vast and diverse dataset spanning a larger amount of time and features and is something that could potentially improve performance as well, but is likely to add even more complexity.

Acknowledgements

Software that I used include the Python libraries Sci-Kit Learn, Numpy, Pandas, XGBoost, and all their dependencies. Material on decision trees, SVM, cross-validation, and best practices from CPTS315 were also used for reference. Data set was retrieved from Kaggle (link below).

Online materials by Sunil Ray gave foundational understanding of boosting. Literature by Dr. Tianqi Chen, Dr. Carlos Guestrin, and Jason Brownlee provided understanding of XGBoost. Work by Ben Gorman provided some guidance as to why stacking was not a big success and largely unused in practice outside of competitions and an article by Alvira Swalin provided some guidance on which metric to optimize upon.

Data set
https://www.kaggle.com/harlfoxem/housesalesprediction

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

1. Bessel, F. W. (2012). The calculation of longitude and latitude from geodesic measurements∗. ArXiv, 1-11. doi:10.1002/asna.201011352
2. Brownlee, J. (2016, September 21). A Gentle Introduction to XGBoost for Applied Machine Learning. Retrieved December 6, 2018, from https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/
3. Chen, T. (2015, September 3). Re: What is the difference between the R gbm (gradient boosting machine) and xgboost (extreme gradient boosting)? [Web log comment]. Retrieved December 6, 2018, from https://www.quora.com/What-is-the-difference-between-the-R-gbm-gradient-boosting-machine-and-xgboost-extreme-gradient-boosting
4. Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. ArXiv, 1-13. doi:arXiv:1603.02754
5. Gorman, B. (2016, December 27). A Kaggler's Guide to Model Stacking in Practice. Retrieved December 6, 2018, from http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/
6. Ray, S. (2015, November 9). Quick Guide to Boosting Algorithms in Machine Learning. Retrieved December 6, 2018, from https://www.analyticsvidhya.com/blog/2015/11/quick-introduction-boosting-algorithms-machine-learning/
7. Swalin, A. (2018, April 07). Choosing the Right Metric for Evaluating Machine Learning Models - Part 1. Retrieved December 6, 2018, from https://medium.com/usf-msds/choosing-the-right-metric-for-machine-learning-models-part-1-a99d7d7414e4
8. Work, S. (2017, August 24). How Loading Time Affects Your Bottom Line. Retrieved December 6, 2018, from https://neilpatel.com/blog/loading-time/