CIS 472/572 Final Project

Mamtaj Akter, Ziyad, Soheil Jamshidi

TOTAL POINTS

29 / 30

QUESTION 1

1 Project 29 / 30

✓ **+ 5 pts** **Presentation: Clear, informative, and thorough.**

+ 4.5 pts Presentation: Pretty good overall, but could have been more polished or thorough.

+ 4 pts Presentation: OK overview, but not clear and/or detailed enough.

+ 3 pts Presentation: Rough and/or very hard to follow.

✓ **+ 10 pts** **Paper: Excellent. Clear, detailed, and thorough, with good description of background and methods, good analysis, and good use of visualizations.**

+ 9 pts Paper: Very good. Could benefit from a little more detail, analysis, and/or clarity.

+ 8 pts Paper: Good. Some good content is present, but parts are hard to read or explanations and analysis are incomplete.

+ 7 pts Paper: Fair. Rough and/or incomplete.

✓ **+ 15 pts** **Methods: Excellent. Careful use of the scientific method, an appropriate variety of well-justified ideas for features and/or models, and insightful analysis into the data and results.**

+ 14 pts Methods: Very good. Could benefit from a little more exploration and analysis.

+ 13 pts Methods: Good. Would benefit from additional exploration and analysis, and/or the methods that were applied may have been slightly incorrect or inappropriate.

+ 12 pts Methods: Mostly good, but with a few holes or errors.

+ 10 pts Methods: Fair. Methods were incomplete or incorrect.

- 0.5 pts Tuning procedures were a little bit unclear.

✓ **- 1 pts** **Tuning procedures were somewhat flawed.**

- 2 pts Tuning procedures were missing or very flawed (e.g., tuning on test data).

Nice selection of methods and analysis of features and results. No discussion of a validation set or cross-validation for tuning. Hyperparameter tuning is very important, not just choosing a reasonable set of defaults -- this can make a big difference in which method works best and how well it works overall.

Predicting the Art of Negotiation

Mamtaj Akter*, Soheil Jamshidi† and Ziyad Alsaeed‡
Computer Science Department,
University of Oregon
Eugene, OR
{*makter2, †jamshidi, ‡zalsaeed}@cs.uoregon.edu

***Abstract*—Understanding the real estate market value before making a once in a lifetime purchase or sell is significantly important. Given a large number of options, personal and professional tastes, and publicly available data sources first-time buyers (or even experienced and dedicated group of people) might have a hard time to make the right decision. Moreover, prices can be heavily dependent on intangible factors (e.g., skills of negotiations) that make it hard even for machine learning models to offer error-free predictions. In our project, we mitigate the mentioned above challenges by carefully analyzing the collected data, cleaning, and enriching dataset with new insightful features. Then, we use the cleaned and enriched data to train different machine learning models and evaluate predictions. After rounds of feature selection and hyper parameter tuning, we finally used 42 features with XGboost ensemble model to get the best result. However, our results suggest that further improvement in model training requires more analysis, experiment, and comprehensive features.**

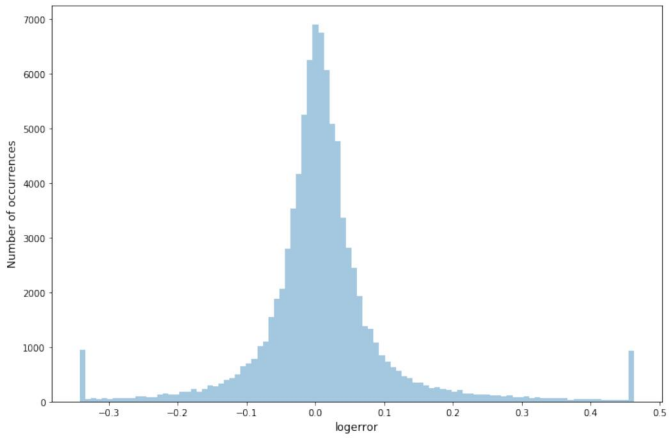

Fig. 1. Logerror distribution of the data collected in year 2016.

I. Introduction

A house purchase is usually the largest investment a person makes during his/her life. Gaining trusted insight into the market before making a purchase or monitoring the value of the owned assets is significantly important. Zillow [7] has been the leading group in evaluating and predicting the real estate market. Using publicly available data about the houses, properties, and transactions history, Zillow developed a combination of machine learning models to predict future property values, which is called Zestimate. Zestimate's median margin of error has improved from 14% in 2006 to only 5% nowadays [2]. The current median margin error is outstandingly good (see Figure-1). In this project, we need to predict the given error.

While Zestimate predicts the sale price of a property, we need to predict how off the Zestimate's prediction is compared to the actual sales transactions ($logerror$). The $logerror$ is defined as follows:

$$logerror = log(Zestimate) - log(SalePrice) \quad (1)$$

In this competition, the goal is to predict the log error in 6-time slots (last three months of 2016 and 2017). However, due to the short time we had, decided to only consider the 2016 data and if we had enough time, work on 2017 data as well, which did not happen. One approach for improving such an accurate predictor is to validate the completeness and correctness of the data for the given labeled records.

The $logerror$ as shown in Figure-1 is significantly low for the majority of the given labeled instances ($\approx 100K$ examples). Predicting the $logerror$ (or essentially improving the prediction of the sale price) requires better features in addition to selecting the right models and parameters values. We are not only trying to predict the $logerror$ based on the properties features (e.g., property size, the number of rooms, quality, etc.) but also based on how skillful the buyers and sellers are in negotiations. Moreover, there are many different factors (features) that can affect the sale price that is beyond the available public data records such as the season of the transaction and the buyers and sellers credit history. Such features are hard (if not impossible) to establish for the given problem. Therefore, complete and correct original features and derived new features from the originals can mitigate the absence of these other factors.

As the problem lands itself in the realm of regression models, finding the right set of features and adding new useful ones are beneficial. To this extent, we are going to try several machine learning models and frameworks that includes Adaptive Boosting, Random Forest, Extremely Randomized Trees, Gradient Boosting and Extreme Gradient Boosting.

The rest of the paper is organized as follows: Section-II is a detailed description of the given problem and data description. In Section-III we describe our approach to solve the key issues of the given problem. Section-IV highlights our experiment and the results we obtained. Next section (Section-V), discusses future work that can be applied to the same problem given the current results. Finally, we conclude in Section-VI.

II. Background

Feature engineering is the most challenging step in this project. The provided original 58 features have a significant

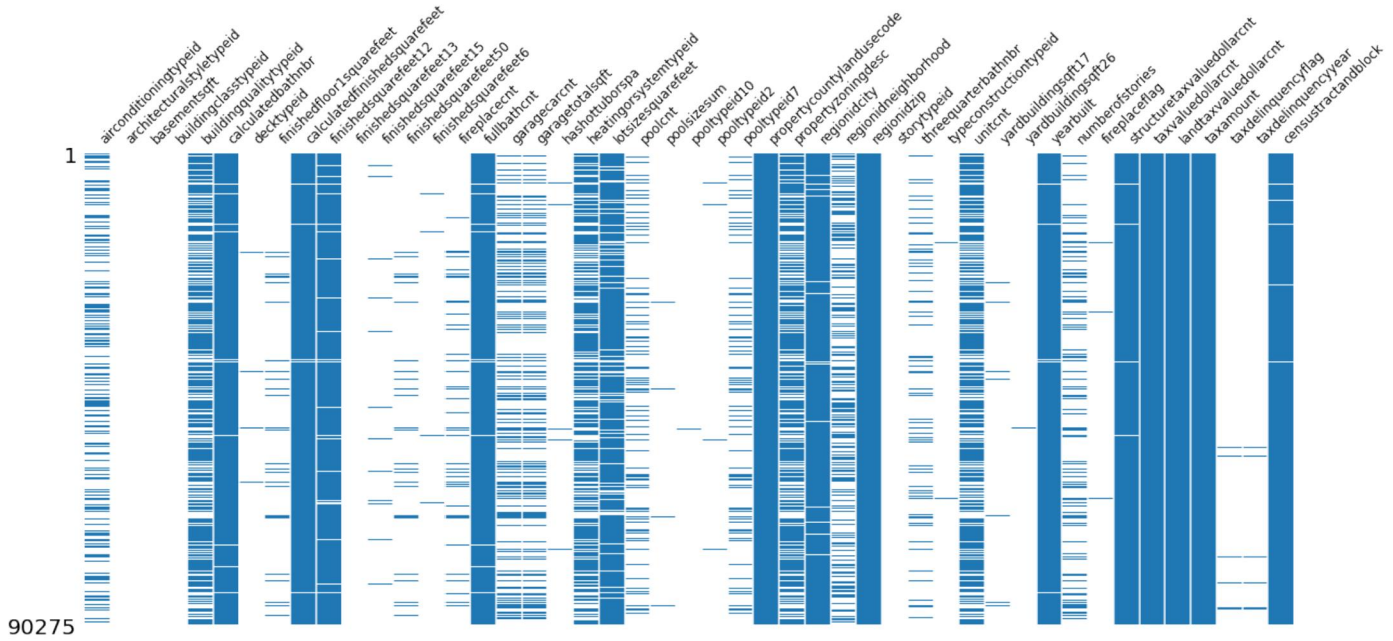

Fig. 2. List of all the features with missing values based on the labeled instances only of the original dataset.

number of missing values as depicted in Figure-2. Each row (Y-axis) represents one of the labeled records and each column (X-axis) stands for one of the features. White gaps in the intersection of rows and columns show that we do not have a value for feature X in record Y. Given the difficulty level and prize amount of this competition, there are not many golden and insightful kernels publicly available, compare to the other Kaggle projects.

We found some strategies for manipulating missing values [6], [1]. Nevertheless, since the margin of error is very small meaning that we are improving an already accurate prediction model, we thought a careful manipulation of each feature with missing values is the best approach. Any unconsidered introduction of new values can degrade the accuracy instead of improving it. Using one of the established techniques for manipulating the missing values can outperform our manual treatment of the features. However, manually treating each feature can allow us to have a better understanding of how to improve the accuracy when we make some predictions.

The given features can be grouped into different types. By using any complete feature (one that has 0% missing values) within each group, we might be able to accurately fill the missing values of other features within the same group. For example, we have different features that are related to property location (e.g., neighborhood, zip code, county, and city id). We can use some of the complete location-related features (such as latitude and longitude) to fill missing values on similar type features but with different representation (e.g., neighborhood id). The major challenges with such process is that (1) we might still have limited number of values to use for predicting the value of missing fields and (2) the computation complexity is significantly large for some features (especially location-related features).

Other values of different groups type might be even more challenging. For example, for some features (e.g., number of bedrooms in a house) it might be impossible to fill the missing values and guarantee that there will be no negative effect on the training and prediction process. For such cases, it is also unreasonable to fill them with average, most common, or median values as they stand as unique features of each property.

In addition to solving the problem of missing values, the original set of features is not the optimal set of features as there is a correlation between the various pair of features. Figure-3 shows the correlation among features where yellow shows the strong positive (e.g., number of pools and pool type) and dark blue represents the strong negative correlation (e.g., garage area and quality).

Furthermore, it is not clear what combination of features might introduce a new insightful feature that improves the accuracy instead of degrading it. Even when using machine learning techniques to calculate the feature importance [4] this might not be always the ultimate solution for different models as our results will show.

An iterative process is the only solution to identify the optimal set of features that boost the accuracy of different models. Another big challenge here is that we have two different factors to tune. First, the set of features to include for prediction. Second, the models' hyper-parameter tuning for each given set of features. Such, iterative process requires a lot of time and precise measurement of the effect of features and hyper-parameter values combination.

III. Methods

A. *Feature Engineering:*

For the given dataset we have 58 features. However, in some cases, most of the values are missing. To fix the issue

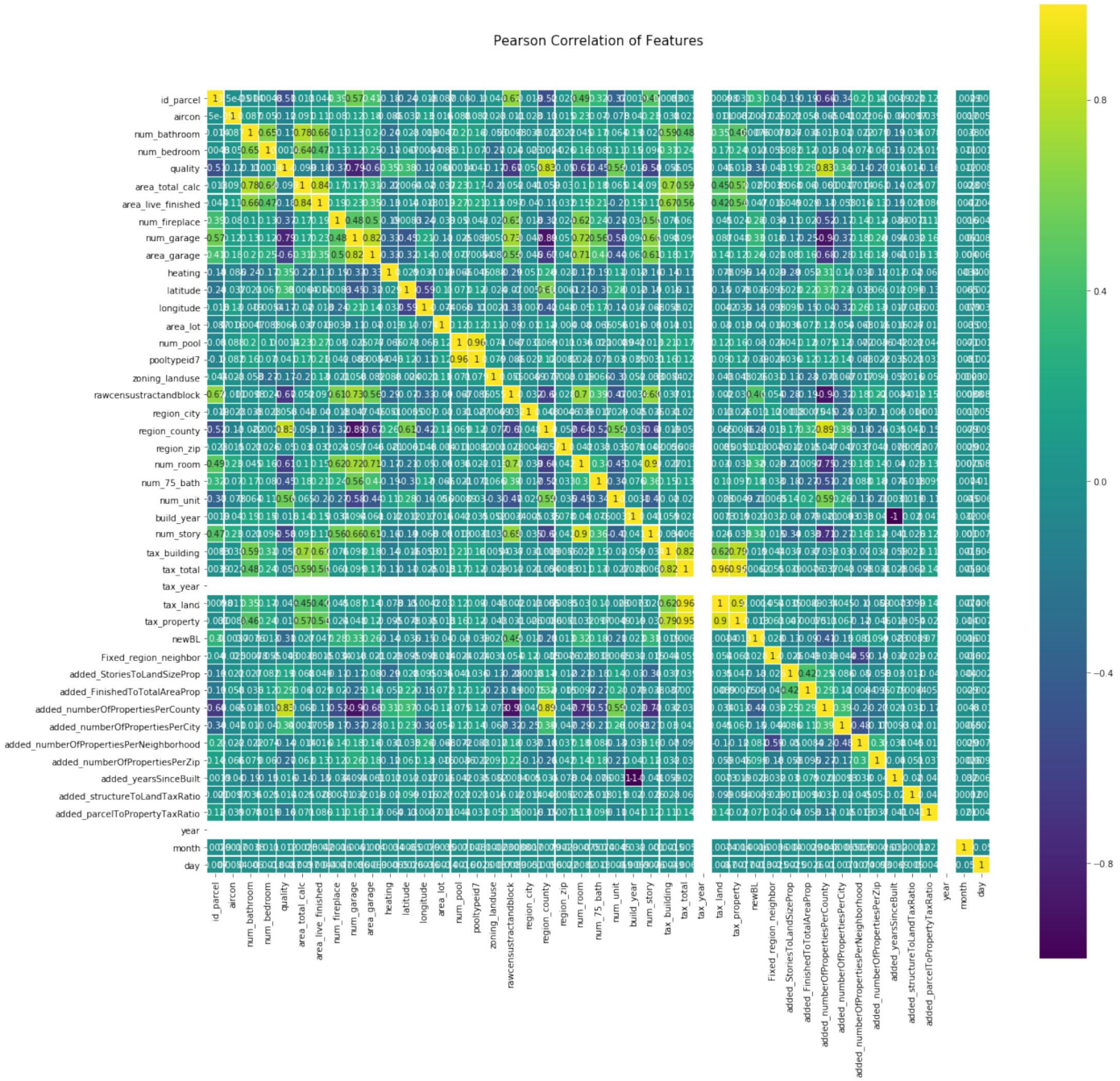

Fig. 3. Value correlation among feature pairs

and possibly introduce new features, we first looked at the fraction of missing values for each feature. Given the $90,275$ records of training data (the set of rows for which we have the *logerror* and transaction date labels) we eliminated all the features that do not satisfy a threshold of the missing data. That is, if the threshold is 50% and we have a feature with 51% null values, then we eliminate that feature. The threshold was set in a way that we can use as many as possible of mostly non-null. For example, if a threshold is set to 95% we may include 33 features, but increasing the threshold slightly to %98 can include 35 features, therefore, we settled on the first choice. The final number of included features we decided to work with is 33 features based on the 95% threshold.

The feature names in the dataset in many cases were confusing as they used codes for them. Therefore we decided to rename the features so we can get a sense of what each feature represents. For example we changed *finishedsquarefeet13* to *area_liveperi_finished* and *finishedsquarefeet15* to *area_total_finished* based on the description provided with the dataset.

Next, we aimed for a deep understanding of the features and filling missing values if possible. In analyzing the features we classified them into two sets in terms of the approach of

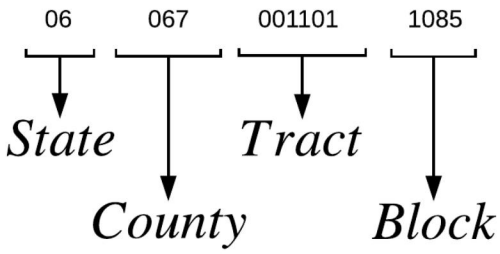

Fig. 4. Interpretation of the feature *rawcensustractandblock*.

filling values for missing data. The first set involves features that have missing data but we do not have any reasonable way to fill the missing fields without increasing the probability of introducing misleading values. For example, features like the number of bedrooms and bathrooms in a house are not possible to be predicted using the information we have from the neighbors. As using techniques like finding the average of a number of bedrooms and bathrooms within a given location range in such cases could mislead the training model into wrong predictions since these are independent and unique features of their architectural design. Therefore, for such features, we filled the missing data with values that are not used so we clearly define the field as missing instead of leaving it as `null`.

The second set of missing data involves features that can be calculated based on other complete features. For example, the neighborhood id is an important feature but involves many missing data (60%). From other features that are complete (has no missing values) such as the latitude, longitude, and census we can calculate the values of the missing fields on the neighborhood id. In this case, we consider all neighbors that are within the half mile of each property given the latitude and longitude of that property and the earth's radius (*haversine* function). Then we fill the missing values with the majority of neighborhood ids.

Such calculation can take a lot of time and resources. To make this resources intensive process viable, we broke down the *rawCensusTractAndBlock* feature based on its definition [5] to help narrow the search scope down. As shown in Figure-4, the Census Tract and block feature is a string that contains the State (2 digits) County (3 digits), tract (6 digits) and the rest is block id. The relation between State, county, tract, and block is depicted in Figure-5

The feature *rawCensusTractAndBlock* helps to reduce the number of records to consider based on different tract codes. In fact, we initially attempted to use the given features to solve for all location related features with missing values. However, we later realized that these zones are not good enough for solving the all missing values because there could be the case where for a given *tract* number there is no recorded neighborhood id. Therefore, combining the census feature along with latitude and longitude helps using larger zones (e.g. *county*) and solve for the majority. After all of the above steps, there are still records with missing values in this group. We can either repeat the process to finally fill all which will lead in low-confidence values and takes time and resource, or we can fill the missing data, not with a value that is not among the values (just like

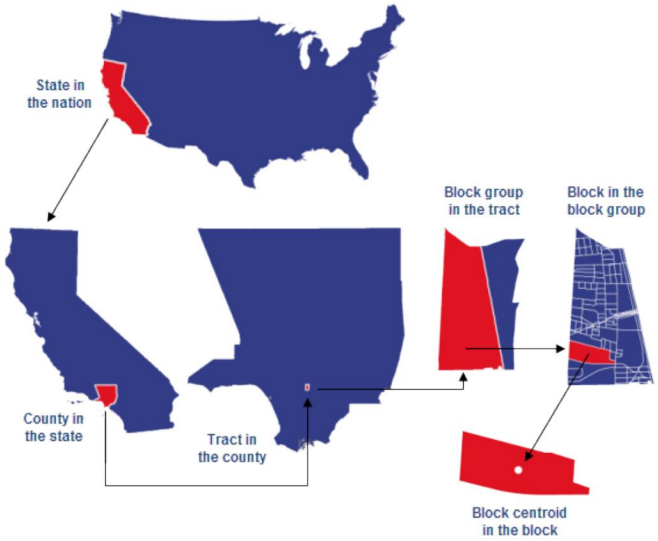

Fig. 5. A guide to understand the county, tract, and block relation[1]

the first group). Due to time constraint, we chose the second option. Following the given methods and steps for the two sets, we ended up with 33 features with no null value.

The given original features allow for creating new insights from different perspectives. Additional features could help the inference models be more accurate and therefore we attempted to create a few more and use them as we tune the model's parameters.

The first two new features we introduced are related the property area. Given that we have the total finished area of the property and the number of stories, we find the proportion of these two values to the total area of the property. These two new features give an indication of how the land area is used vertically and horizontally. Another perspective we could gain from original features is how densely populated is the area in which the property resides. From the county, city, neighborhood, and zip codes, we count the number of properties associated with each area type id. Such new features give a new perspective on how populated the area is. Also, given that we have the year in which the property is built, we calculate the age of the property as of the year 2016. As the year 2016 is the year in which we are trying to predict the $logerror$ of prices and it is the year in which the data was collected. Finally, we created two additional features based on the different tax-related features. One defines the structure to land tax ratio and the other defines the parcel to property total tax assessment ratio. This gives a total of 9 new features.

From the original features we didn't try to find any new features based on the property description (e.g., number of bedrooms, number of bathrooms, etc) as these are usually incomplete and there was no appropriate method to fill the missing data. Therefore, creating new features based on partial data can be misleading. Moreover, we thought that the location related original features (e.g., latitude, longitude, and county) are sufficient enough (given that we filled the missing values correctly) and no further insight is necessary.

The final total number of features we are to work with is

42 features.

B. Model Selection

We decide to apply stacking ensemble method Extreme-Gradient Boosting (XGBoost) as it has recently been vastly used in real-life machine learning competitions. XGBoost is a kind of gradient boosting especially designed for higher speed and high performance. The base models we use for the first-level prediction are as follows:

- *Support Vector Machine (SVM)*: SVM is a good choice when datasets have numerous features. Since we also have 43 features in total, we pick SVM in first-level training.
- *Random Forest (RF)*: RF works efficiently with large datasets and correctly estimates of what features are important for the classification. RF also maintains the accuracy when a large proportion of the data are missing which is a perfect fit for our project.
- *Extra Trees (ET)*: For real-world problems with a large dataset like our project (90275 labeled data), Extra Trees are perfectly applicable as the performance of Extra trees is comparable to the random forest, sometimes a bit better. At each step of the extra trees, the entire sample is used and decision boundaries are picked randomly which makes ET more computationally efficient.
- *Gradient Boosting (GBT)*: We apply GBT for its two special characteristics Faster and better performance. Generally, it builds one tree at a time and each newly built tree helps to correct the errors that were made by the immediate previous tree and thus, the model becomes more robust.

IV. Experiments

In this section, we describe the experiments and analysis that we conducted and the results we found.

A. Experimental Procedure:

We take 80% of the labeled data for training and then test the results with the remaining 20% of the data. So, we have $72,428$ training examples and $18,107$ test examples finally to experiment on. We make the test data by setting the year and month according to each time period we are being asked to predict the *logerror* for and by assigning a random day.

We apply the method of ensembling models, in particular the variant we use here of ensembling known as Stacking. Stacking uses the predictions as a base of a few basic machine learning models (here, we use Random Forest, Extra Trees, Support Vector Machine and Gradient Boosting Regression) and then uses another model (XGBoost) at the second-level to predict the output from the earlier first-level predictions.

B. Tuning Parameters:

In case of hyper-parameter tuning, we set the parameters of Random Forest and Extra Trees such that the model can be faster and also give a good accuracy. So, we limit the max-features (number of features to consider when looking for the best split) as the `sqrt(n_features)`. And we also do not minimize the number of total trees built in the forest and so we set `n_estimatores = 500`. We keep `max_depth` and `min_samples_leaf` (The minimum number of samples required to be at a leaf node) as only 6 and 2 respectively for Random Forest. All the parameters were same in Extra Trees except the `max_depth` value which is 8. For Support Vector Machine, we take a moderately minimal penalty parameter $C = 0.025$. And the gradient-boosting parameters are as follows: `n_estimators = 500`, `learning_rate = 0.1` and `max_depth = 1`.

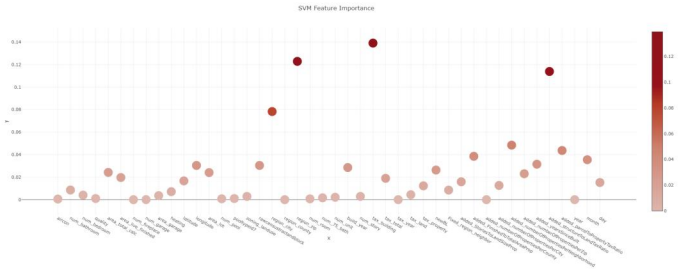

Fig. 6. Feature Importance in SVM Regressor.

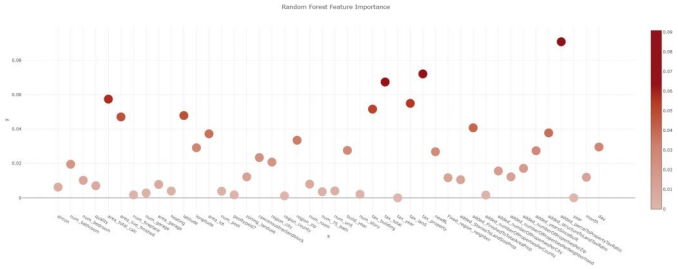

Fig. 7. Feature Importance in Random Forest Regressor.

In the second-level ensemble method, we set the parameters of XGBoost as accordingly: `learning_rate = 0.02`, `n_estimators = 2000` (we increased the total number of trees can be built by four times), `max_depth = 4` and `min_child_weight = 2`.

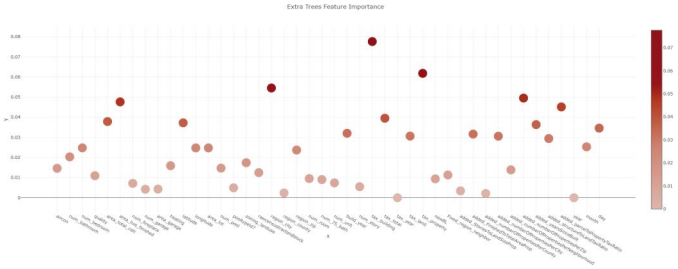

Fig. 8. Feature Importance in Extra Trees Regressor.

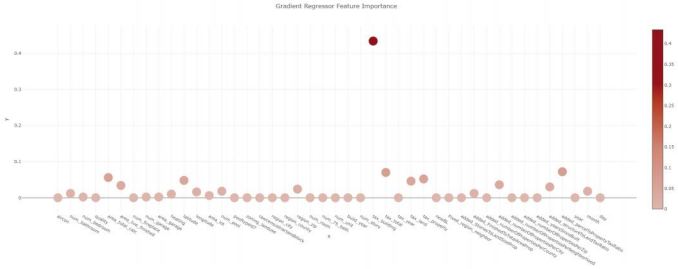

Fig. 9. Feature Importance in Gradient Boosting Regressor.

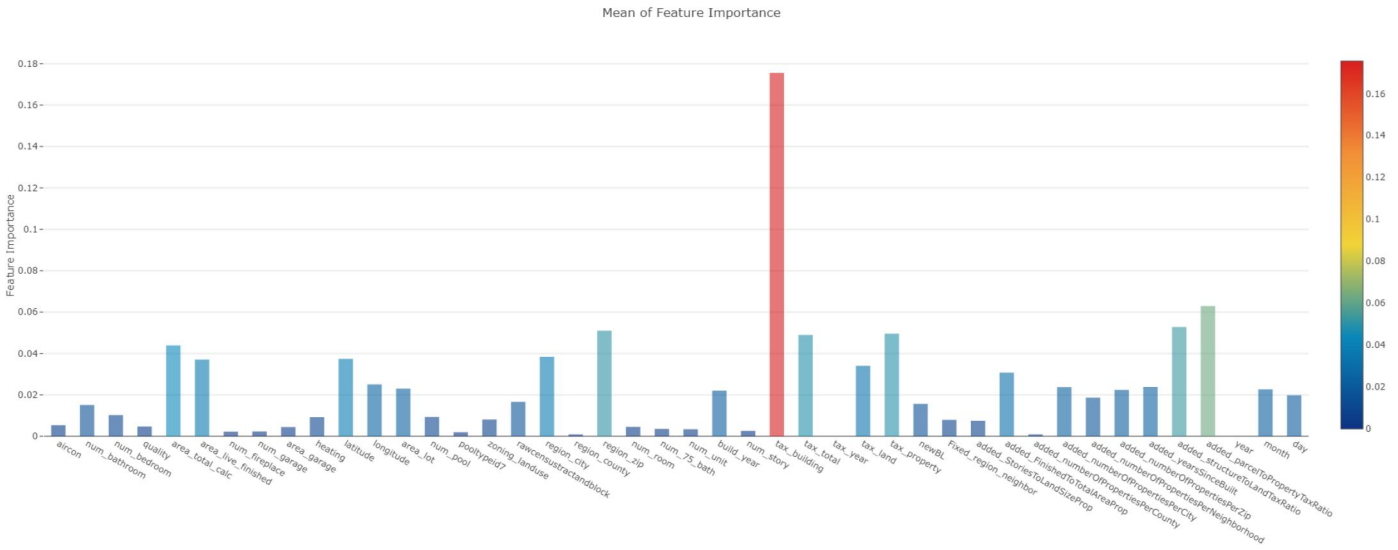

Fig. 10. Mean Feature Importance.

After the first-level training, using the above discussed four machine learning models, we get 4 sets of the test set with predicted labels.Then we analyze the base predictions in two ways (1) feature importance for the next level ensemble and (2) the correlation among the predictions.

C. Analysis of Feature Importance:

Generally, feature importance provides a score that indicates how valuable each of the features was for the predictions within the model. The more an attribute is used to make the key predictions with a model, the higher its importance. Therefore, feature importance graph helps us to select the features that will be given priority in the second-level ensemble method. So, we analyze the important features in each of the regressors we used in the first-level ensemble. Figures-6 7, 8 and 9 plot the importance percentages per each of the predictions. Figure-6 shows that the leading features based on SVM are *added_StructureToLandTaxRatio*, *Tax-building*, *region_zip* and *region_city*. In Figure-7, we get the *added_parcelToPropertyTaxRatio*, *newBL* and *tax-total* as the three most important feature that had over 60% importance on the output, whereas in Figure-8 we get *tax_property* and *tax_building* makes above 60% of importance. Most notably, the gradient regressor (Figure-9) is mostly dependent on only one feature *tax_building*. The feature importance vector is then averaged across all of the regressors (Figure-10) to analyze the mean feature-importance and we get only two leading features which are above 60% important on average.

D. Correlation of Base Predictions:

The Figure-11 shows the correlation among the base predictions that we get as the output from the first-level methods. This correlation graph depicts how our regressors correspond with each others' output. Here, we find an interesting correlation between each pair of the regressors. We get the highest correlation between the predictions of Gradient Boosting and Random Forest, whereas the pair of Support Vector Machine and Random Forest Regressor does not correlate at all.

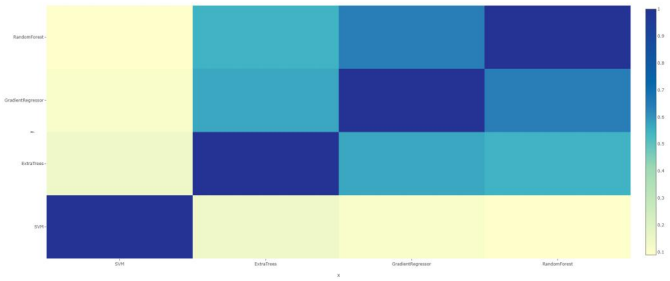

Fig. 11. Correlation among the Regressors.

After trying different methods, also using the codes from [3] kernel, we used XGBoost stacking ensemble method to benefit from advantages of four base predictors as they were not accurate enough individually. We have uploaded the results of applying XGboost on test data on Kaggle. The best final score we got was 0.0774231 in private and 0.0663010 in public section. Which is reasonably close to the top scores with 0.0740861 and 0.0631885 in private and public, respectively. However, since the competition is tight that score puts us among top 70% of the participants.

V. Future Work

If we had more time, experimenting the recursive filling of location-related missing values and effect of looping into feature, model, and hyper-parameter selection and tuning were among our tasks. Applying the same process of data analysis, cleaning, feature engineering, and model selection on the second half of the data related to transaction in 2017 was another part of the task that we were not able to complete given the time constraints. Also, creating a more accurate zoning that would help us safely fill the remaining missed data in some location-related features is among the task we wished to tackle.

VI. Conclusion

Improving the Zestimator $logerror$ sounds like a challenging project from the beginning. We were involved in data analysis, data cleaning, feature engineering, model selection on a large scale real-world dataset with a large number of missing values. Lack of domain knowledge, time management, unknown and hidden risks of the projects including unexpected issues in coding, resources, and results were among our challenges that added to our experience and skill set.

After filling the missing values using the proper values (based on neighbors and location in some cases) we added 9 insightful features and used 4 base models in XGboost ensemble. In the end, we got a score that put us among the top 60% of the competition. However, given the time limit, amount of prize ($1.2M), and the difficulty level of the competition it meets our expectations.

We can expand this project by using more data sources that are publicly available (like local crime reports in each neighborhood), applying different methods for handling missing values (like recursively filling missing value using majority voting) and also working on the data records they have published from 2017 transactions.

References

[1] Z. Ghahramani, Z. Ghahramani, and M. I. Jordan. Supervised learning from incomplete data via an em approach. *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 6*, 6:120–127, 1994.

[2] Kaggle. Zillow prize: Zillow's home value prediction (zestimate). https://www.kaggle.com/c/zillow-prize-1.

[3] Kaggle. Zillow, stacking ensemble of regressors. https://www.kaggle.com/jamesdhope/zillow-ensemble-of-regressors-0-065.

[4] K. Kira and L. A. Rendell. A practical approach to feature selection. In D. Sleeman, , and P. Edwards, editors, *Machine Learning Proceedings 1992*, pages 249 – 256. Morgan Kaufmann, San Francisco (CA), 1992.

[5] ProximityOne. Census blocks and block codes. http://proximityone.com/geo_blocks.htm.

[6] A. J. Smola, A. J. Smola, S. V. N. Vishwanathan, and T. Hofmann. Kernel methods for missing variables. *IN PROCEEDINGS OF THE TENTH INTERNATIONAL WORKSHOP ON ARTIFICIAL INTELLIGENCE AND STATISTICS*, pages 325–332, 2005.

[7] Zillow. Zillow. https://www.zillow.com/.

1 Project 29 / 30

✓ **+ 5 pts Presentation: Clear, informative, and thorough.**

+ 4.5 pts Presentation: Pretty good overall, but could have been more polished or thorough.

+ 4 pts Presentation: OK overview, but not clear and/or detailed enough.

+ 3 pts Presentation: Rough and/or very hard to follow.

✓ **+ 10 pts Paper: Excellent. Clear, detailed, and thorough, with good description of background and methods, good analysis, and good use of visualizations.**

+ 9 pts Paper: Very good. Could benefit from a little more detail, analysis, and/or clarity.

+ 8 pts Paper: Good. Some good content is present, but parts are hard to read or explanations and analysis are incomplete.

+ 7 pts Paper: Fair. Rough and/or incomplete.

✓ **+ 15 pts Methods: Excellent. Careful use of the scientific method, an appropriate variety of well-justified ideas for features and/or models, and insightful analysis into the data and results.**

+ 14 pts Methods: Very good. Could benefit from a little more exploration and analysis.

+ 13 pts Methods: Good. Would benefit from additional exploration and analysis, and/or the methods that were applied may have been slightly incorrect or inappropriate.

+ 12 pts Methods: Mostly good, but with a few holes or errors.

+ 10 pts Methods: Fair. Methods were incomplete or incorrect.

- 0.5 pts Tuning procedures were a little bit unclear.

✓ **- 1 pts Tuning procedures were somewhat flawed.**

- 2 pts Tuning procedures were missing or very flawed (e.g., tuning on test data).

Nice selection of methods and analysis of features and results. No discussion of a validation set or cross-validation for tuning. Hyperparameter tuning is very important, not just choosing a reasonable set of defaults -- this can make a big difference in which method works best and how well it works overall.