2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems

Predicting Zillow Estimation Error Using Linear Regression and Gradient Boosting

Darshan Sangani[1], Kelby Erickson[2], and Mohammad Al Hasan[3]

***Abstract*— Owning property is one of the most important investments that a person can make in their lifetime. Therefore, being able to accurately know the real-time value of any property is crucial for making wise sales and purchases. Since the online real estate database company Zillow first developed a machine learning system to predict property sale prices in real time, it has continually worked to improve the accuracy of this prediction mechanism.**

In this paper, we describe our work to decrease the error of Zillow's price estimation by examining the effectiveness of several machine learning models and techniques at making property related forecasts. Specifically, we used property data to train linear regression and gradient boosting models with which we then made predictions about other properties. For the gradient boosting model, we used grid search to fine-tune the model's hyperparameters and observed the contribution of such tuning to the model's accuracy. Finally, we examined the effectiveness of several data preprocessing techniques, including the novel approach of treating time as a property feature.

***Index Terms*— Machine learning, linear regression, gradient boosting, real estate, Zillow**

I. INTRODUCTION

As the leading real estate database company, Zillow[4] has developed the most accurate property value forecasting algorithm to date. Its proprietary home value forecasting method, which provides house price estimates called "Zestimates," is available online for free use. However, no machine learning prediction mechanism is exact, so Zillow is always seeking to improve the accuracy of its algorithm. As a part of this effort, Zillow has hosted a competition called Zillow Prize[5] [1] on the machine learning competition website Kaggle which will reward the contestant who creates the best mechanism for estimating property values.

Zillow provides all contestants in this competition with a set of input data and a set of output data. The input data consists of various properties, each with a set of values for various features. The output data consists of a subset of the input properties, each accompanied by the decimal logarithm of the corresponding Zestimate's error (log error) for certain dates. The set of input-output pairs serves as training data that each contestant uses to develop a prediction model. Such a model is then used to predict the log error for each input property (the testing data) for six different months. The contestant whose estimates most closely match Zillow's corresponding estimates wins the competition.

In this paper, we describe the machine learning approaches we took while competing in this competition and the effectiveness of each at predicting the log error. First, we cover related research and several models that have already been used to make predictions about real estate properties. Then we discuss the specific learning algorithms and preprocessing techniques that we used to develop prediction models, including our novel preprocessing approach of treating time as a feature in the input matrix. Following this, we describe the property data provided by Zillow through the Zillow Prize competition and the criterion we used to evaluate the accuracy of various techniques. Finally, we showcase our results and discuss their significance.

II. RELATED WORK

In this section, we discuss previous machine learning research that dealt with making predictions about the real estate market. In order to provide an overview of the various machine learning techniques used in this field, we focus on those techniques used in the related research that are different from our own techniques.

In a recent work, Bhuiyan et al. [2] proposed a mechanism to predict how soon a house will be sold once it is placed on the market. They used housing data from five cities in Indianapolis obtained from the online residential real estate site Trulia.[6] The data set consists of several house features such as the bedroom count, bathroom count, and number of fireplaces, etc. Using several of those features, they computed the survival probability of a house, which plays a key role in predicting how long it will take to sell the house because the main model used for prediction was based on survival regression. Compared to this research, we are predicting something different: Zillow log error. However, we have a similar feature selection process as the one used in this research.

Another work done in this field by Lim et al. [3] involved predicting condominium prices. Condominiums are a special type of private housing which have built-in facilities such as a clubhouse, barbecue area, gymnasium, and swimming pool. For this project, the researchers predicted condominium prices based not only on housing features and facilities but also on variables such as the gross domestic product, consumer price index, prime lending rate, and real interest rate. They mainly used three machine learning models based

[1]D. Sangani is with the Department of Computer Science, Indiana University–Purdue University Indianapolis, 723 W. Michigan Street, Indianapolis, IN 46202, USA

[2]K. Erickson is with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712, USA

[3]M. Hasan is with the Department of Computer Science, Indiana University–Purdue University Indianapolis, 723 W. Michigan Street, Indianapolis, IN 46202, USA

[4]zillow.com

[5]kaggle.com/c/zillow-prize-1

[6]trulia.com

2155-6814/17 $31.00 © 2017 IEEE
DOI 10.1109/MASS.2017.88

on artificial neural network (ANN), autoregressive integrated moving average (ARIMA), and multiple regression analysis (MRA) to predict condominium price indices (CPI) and condominium asking prices (CAP). These researchers concluded that the ANN model performs better than the ARIMA one at predicting CPI and better than the MRA one at predicting CAP. Also, they treated time as an independent variable separate from the input matrix while we took a different approach by treating time as a feature in our input matrix.

III. LEARNING TECHNIQUES

The estimation problem posed by the Zillow Prize competition is an example of supervised learning, a subset of machine learning that seeks to develop a prediction function using both input data and its corresponding output data. In general, given a data set with n instances and m features $D = \{X, Y\}$ $(X = \{x_{i1}, ..., x_{im}\}_{i=1}^{n},\ Y = \{y_i\}_{i=1}^{n})$, the goal of supervised learning is to find a function F such that $F(x_i)$ predicts y_i [4]. For our particular problem, the goal is to find a function that takes a property-by-feature matrix and returns a matrix of the log errors associated with those properties depending on the month.

In this section, we discuss the two learning algorithms of linear regression and gradient boosting that we used to develop our prediction models. We also describe the technique of grid search that we used to optimize the hyperparameters of the gradient boosting algorithm.

A. Linear Regression

The simplest machine learning technique that we used is linear regression [5] [6], which develops the prediction function F as a linear function of the input vector x. It does this by finding an intercept ω_0 and coefficient vector $\omega = (\omega_1, ..., \omega_m)$ such that

$$F(x) = \omega_0 + \omega^T x = \omega_0 + \omega_1 x_1 + ... + \omega_m x_m$$

This is accomplished through a least-squares approach, in which the sum of squared differences between predicted values and actual values is minimized. Mathematically, linear regression solves a problem of the following form:

$$\min_{\omega} ||X\omega - Y||^2$$

We chose to use linear regression as our first model-generating technique due to its simplicity and wide-spread use in the field of machine learning.

B. Gradient Boosting

Gradient boosting [7] [8] [9] is an algorithm that works on the principle of ensemble learning, in which an ensemble of decision trees is built and the predictions of individual trees are summed in order to calculate the overall prediction $D(x)$. This is represented mathematically as

$$D(x) = d_{tree1}(x) + d_{tree2}(x) + ...$$

Once one tree has been built, the next tree is built in order to cover the discrepancy between the target function $F(x)$ and the current ensemble prediction. For example, if the ensemble currently has two trees such that

$$D(x) = d_{tree1}(x) + d_{tree2}(x)$$

then the next tree in the ensemble (tree 3) will be trained in such a way that it complements the already existing trees and minimizes the training error of the ensemble [10]. Ideally, the next tree should be such that

$$D(x) + d_{tree3}(x) = F(x)$$

To refine the final prediction, the algorithm trains trees to reconstruct the difference between the target function and the current prediction of the ensemble. This difference is called the residual:

$$R(x) = F(x) - D(x)$$

If a decision tree completely reconstructs $R(x)$, the whole ensemble makes predictions without error. In practice this never occurs, so the iterative process of building trees continues.

A decision tree is a tree-like graph of decisions. It is a fairly simple classifier which splits a space of features into regions by applying conditional splitting (e.g., number of bedrooms < 4). Gradient boosting merges a set of weak learners (individual decision trees that alone are poor predictors) and delivers improved prediction accuracy by leveraging the entire ensemble. For any tree, the prediction outcomes of the tree are weighted based on the outcomes of the previous tree. Also, the more accurate an outcome is, the more it is weighted. When the entire ensemble of trees is used to make a single prediction, each tree will contribute differently depending on the weightings of its outcomes.

In addition to the traditional gradient boosting algorithm, there is a more optimized version of the algorithm called XGBoost (extreme gradient boosting) that tends to scale better than the traditional algorithm [4]. We developed one of our prediction models using this to see how XGBoost performs relative to normal gradient boosting.

C. Hyperparameter Tuning

While performing gradient boosting on a data set, one problem to avoid is overfitting the model [11]. Overfitting occurs when a model learns too much from random fluctuations and noise that are unique to the training data set. As a result, the model will make accurate predictions based on the training input data, but will make poor predictions when given any other input. In other words, an overfit model does not generalize well. To avoid overfitting, we worked to fine-tune the gradient boosting hyperparameters, which are parameters not learned directly within a model.

In order to do this, we used grid search with cross-validation, an algorithm that exhaustively considers all parameter combinations given to it as input. First we specified several different values for each of three gradient boosting hyperparameters (the number of estimators, the learning rate, and the maximum depth). Then the grid search algorithm went through all specified values for each parameter in order to find the optimal values to use in our boosting model.

IV. PREPROCESSING TECHNIQUES

In this section, we describe the various preprocessing techniques we performed on our data set in order to make it easier for the linear regression and gradient boosting algorithms to learn from the data and develop more accurate models.

A. Flattening categorical features into binary ones

A categorical feature is one that can take on one of a limited number of values (usually integers), each of which corresponds to a different category. For example, the Zillow Prize data set contains the categorical feature $StoryTypeID$ which can take on values such as 1 if the property has an attic and a basement, 2 if it has only an attic, 3 if it is bi-level with an attic and a basement, etc.

Since linear regression performs poorly when dealing with categorical features, we flattened all of the categorical features in the Zillow Prize data set into binary ones. A binary feature is one that can take on either 0 for false or 1 for true. Continuing with the previous example, $StoryTypeID$ can take on 35 different categorical values, so we flattened it into 35 different binary subfeatures: $StoryTypeID_1, ..., StoryTypeID_{35}$. Each of these could take on either 0 or 1, and for each property, only one of the subfeatures of a particular feature could take on 1 (since if one of a set of mutually exclusive conditions is true, the others must be false). For example, if a property has an attic and a basement, the subfeature $StoryTypeID_1$ would take on 1 while $StoryTypeID_2, ..., StoryTypeID_{35}$ would each take on 0. In general, this preprocessing technique is necessary for linear regression to generate an accurate model and so was the first that we performed.

B. Normalization

Normalization [12] is another preprocessing technique used to give all features a uniform scale. Since each feature in the Zillow Prize data set has a different scale and linear regression tends to deal better with data that has the same scale for each feature, we decided to normalize the data and observe the effect on our linear regression model.

To normalize feature j, we first calculated the average μ_j and standard deviation σ_j of the feature. Then we replaced each value for that feature $\{x_{ij}\}_{i=1}^{n}$ with

$$\frac{x_{ij} - \mu_j}{\sigma_j}$$

Such normalization was intended to make the linear regression algorithm treat each feature equally rather than give more weight to certain features simply due to a difference in scale.

C. Dimensionality reduction

Another preprocessing technique we used is dimensionality reduction, which decreases the total number of features in a data set by combining correlated features. To accomplish this, we applied principal component analysis (PCA) [13], a statistical procedure that converts a set of features that may be linearly correlated into a set of principal components that are linearly uncorrelated. For example, the Zillow Prize data set contains the features $PoolCnt$ (the number of pools on the property) and $PoolSizeSum$ (the total square footage of pools on the property). Since $PoolCnt$ and $PoolSizeSum$ intuitively have a positive linear correlation, PCA is likely to combine them into a single feature. Overall, we used PCA to eliminate noise in the data set and increase the variance between property features, expecting such changes to improve the accuracy of our gradient boosting model.

D. Treating time as a feature

A final preprocessing technique we used that has not been used before in machine learning involves treating time as a property feature within our data set. Since the Zillow Prize competition requires contestants to make predictions for six different months (October, November, and December of 2016 and 2017), we added the categorical feature $Month$, representing the month a property was sold. $Month$ can take on any integer from 1 to 24 (1 corresponding to January 2016, 2 to February 2016, and so on until 24, which corresponds to December 2017).

Given any developed prediction model, in order to make predictions for a certain month, we would first append the $Month$ feature to the input testing matrix (consisting of all properties for which we desired to make a prediction). Then we would set the $Month$ value for each property equal to the integer that we had assigned to the desired month as described in the previous paragraph. This extended version of the input testing matrix would then be used as input to the model so as to output the log error for each property for that month.

One advantage of this technique is that it allowed us to incorporate multiple sales of a property into our input matrix. For example, if a property were sold twice, it could be treated as two separate instances in our input training matrix, each with the exact same values for every feature except $Month$. Since the number of properties that were sold more than once is small compared to the total number of properties sold (less than 0.14%), we did not have enough data on which to perform the more traditional approach of time series analysis.

V. DATA SET AND EVALUATION CRITERION

A. Data Set

All of the data used was provided by the Zillow Prize competition [1] in the form of two comma-separated values (CSV) files, one with input data and one with output data. The input CSV consists of 2,985,217 parcel identifiers along the vertical axis, each corresponding to a different property located in one of three counties (Los Angeles, Orange, and Ventura, California) in 2016. Along the horizontal axis are 58 property features such as zip code and tax amount. The output CSV, like the input CSV, has parcel identifiers along its vertical axis. However, only 90,275 properties are provided as output, all of which are members of the input data. For each property, the log error for a certain sale and

the date of that sale are provided. Log error is defined as

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

where $Zestimate$ is Zillow's estimate of the sale price and $SalePrice$ is the actual sale price.

B. Evaluation Criterion

The accuracy of our estimate was evaluated using mean absolute error (MAE), which is a measure of the average absolute difference between predicted values and actual values. Commonly used as a measure of forecast error, it is defined as

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

Where

$$AE = |e_i| = |y_i - \hat{y_i}|$$

$$Actual = y_i$$

$$Predicted = \hat{y_i}$$

Whenever a contestant submits a prediction to the Zillow Prize competition, Kaggle automatically calculates the MAE between the contestants prediction of the log errors and the actual log errors. Since up to five submissions are allowed every day, we used the MAE values calculated by Kaggle to gauge the effectiveness of the various prediction models and techniques that we used.

VI. RESULTS AND DISCUSSION

The Zillow Prize data set was used to train and test various prediction models based on linear regression (LR) and gradient boosting (GB). The specific LR and GB models we used are those included in scikit-learn[7], a free machine learning library designed for the Python programming language. Before applying either of these two model-generating algorithms to the data, we used various preprocessing techniques to make the data easier to learn from. For all models, we used our novel approach of treating time as a property feature.

TABLE I: Performance of the Prediction Models

Model	MAE
LR	0.1934376
LR (with normalization)	64×10^6
GB (with LS loss)	0.0652937
GB (with LAD loss)	0.0645612
GB (with LAD loss & PCA)	0.0650162
GB (with LAD loss & grid search)	**0.0644700**
XGBoost	0.0648888

Note: Parentheses indicate techniques that were used in addition to the base model, indicated by a lack of parentheses. For example, the first table entry is LR without normalization, and the second is LR with normalization.

Each model and its lowest MAE (since some models were built more than once using different parameter values) are displayed in Table 1.

[7]scikit-learn.org

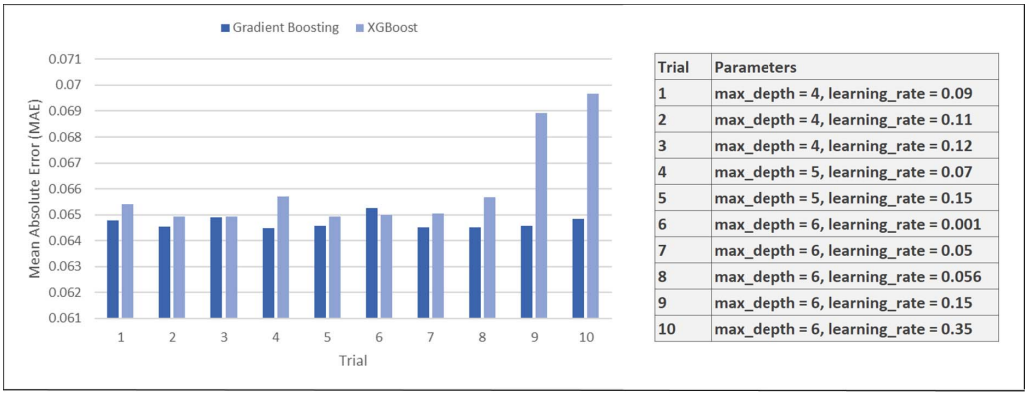
We trained two different models based on linear regression, the first before performing normalization and the second after performing normalization. Before training either of these two models, we performed the preprocessing technique of flattening categorical features into binary ones.

Using gradient boosting, we trained five different models. One of these models was built through XGBoost. The rest were trained by the traditional gradient boosting algorithm. Of these remaining four models, one used the least squares (LS) loss function; the other three used the least absolute deviation (LAD) loss function. Of these three models, one trained on data that had been preprocessed using principal component analysis (PCA), and one used grid search to optimize the hyperparameters of the gradient boosting algorithm.

As can be seen in Table 1, all of the gradient boosting models outperformed the linear regression ones. This was expected considering the fact that gradient boosting develops an ensemble of decision trees while linear regression simply finds a line of best fit. Among the gradient boosting models, those using the LAD loss function were more accurate than those using the LS loss function. Since LAD sums absolute errors while LS sums the squares of the absolute errors, LS is effected more by outliers than is LAD. Since using LAD worked better than using LS, we can infer that the Zillow Prize data set contains numerous outliers.

One unusual find of our research is that the preprocessing technique of normalization (generally considered to improve prediction accuracy) decreased the accuracy of our prediction mechanism. This is clearly seen when we compare our two linear regression models: the one that used normalization has an MAE that is over 300 million times greater than that of the one that did not use normalization. Since the normalization of a certain feature is dependent on that feature's mean and standard deviation and since outliers have a significant effect on those values, the presence of outliers in a data set sometimes causes normalization to worsen scaling rather than improve it. As demonstrated by the advantage of using LAD over LS, the Zillow Prize data set likely contains outliers, so it is not surprising that such outliers would skew our normalization of the data set.

Another unexpected result of our work is that the prediction models built by normal gradient boosting using an LAD loss function tended to forecast log error more accurately than those built by XGBoost, which is similar to normal gradient boosting except for a few additional optimizations. XGBoost and gradient boosting have two key parameters in common: max_depth, the maximum depth of each decision tree, and $learning\_rate$, the rate at which the optimal splitting point at a tree node is found. Taking advantage of this, we built an XGBoost model and gradient boosting model for each of ten different randomly generated combinations of max_depth and $learning\_rate$. For all of the gradient boosting models, we used an LAD loss function. As displayed in Figure 1, all but one of the models built by gradient boosting have a lower error than those built by XGBoost using the same values for max_depth and $learning\_rate$.



Trial	Parameters
1	max_depth = 4, learning_rate = 0.09
2	max_depth = 4, learning_rate = 0.11
3	max_depth = 4, learning_rate = 0.12
4	max_depth = 5, learning_rate = 0.07
5	max_depth = 5, learning_rate = 0.15
6	max_depth = 6, learning_rate = 0.001
7	max_depth = 6, learning_rate = 0.05
8	max_depth = 6, learning_rate = 0.056
9	max_depth = 6, learning_rate = 0.15
10	max_depth = 6, learning_rate = 0.35

Fig. 1: Comparison of XGBoost and gradient boosting for various combinations of max_depth and $learning\_rate$

As for our treatment of time as a property feature, doing so yielded different log errors for different months for a given property when we used linear regression but not when we used gradient boosting. Since linear regression does not have a mechanism for weighing different features over others while gradient boosting does, it is possible that the $Month$ feature always had little relevance to the overall prediction of log error but that only the gradient boosting model was able to recognize this. It could also be a result of there being an insufficient number of properties that were sold more than once and therefore not enough data to make accurate time-series predictions of any sort. Finally, even though there is an obvious relationship between time and property values, there is not an obvious relationship between time and log error, so it should not be surprising if for a given property, there is little or no variation between log errors for different months.

Overall, the best performing model is the one generated by gradient boosting combined with grid search. Since the goal of grid search is to test different values for various parameters of the gradient boosting algorithm in order to find the optimal set of parameter values, it makes sense that using it would result in a more accurate prediction model.

VII. CONCLUSIONS

Since buying a house is the largest purchase that the average person makes in their life, being able to accurately predict the real-time value of any property is crucial. In order to improve Zillow's mechanism for estimating property values, we investigated the effectiveness of several linear regression and gradient boosting models, each of which used a different combination of machine learning techniques. Overall, we found that for this particular problem, linear regression is outperformed by gradient boosting, particularly gradient boosting that uses the least absolute deviation loss function and grid search parameter optimization. We also showed that normalizing the Zillow Prize data set is not effective, likely due to outliers. Finally, our novel approach of treating time as a property feature demonstrated that time is not a significant indicator of log error.

ACKNOWLEDGMENTS

This research was made possible with the support of the Indiana University–Purdue University Indianapolis Department of Computer Information and Information Science, as well as through funding from the National Science Foundation (grant no: REU-1560020) and the United States Department of Defense. The authors would like to thank their mentor Dr. Mohammad Al Hasan, as well as Dr. Feng Li, Dr. Eugenia Fernandez, and Sheila Walter for their support.

REFERENCES

[1] Zillow, "Zillow Prize: Zillow's Home Value Prediction (Zestimate)," 2017.

[2] M. Bhuiyan and M. A. Hasan, "Waiting to be sold: Prediction of time-dependent house selling probability," in *Proc. 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 468–477, Oct 2016.

[3] W. T. Lim, L. Wang, Y. Wang, and Q. Chang, "Housing price prediction using neural networks," in *Proc. 2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 2016.

[4] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, 2016.

[5] X. Yan and X. G. Su, *Linear Regression Analysis*. World Scientific Publishing, 2009.

[6] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to Linear Regression Analysis*. John Wiley & Sons, 5th ed., 2012.

[7] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, pp. 1189–1232, 2000.

[8] J. H. Friedman, "Stochastic gradient boosting," *Comput. Stat. Data Anal.*, vol. 38, pp. 367–378, Feb. 2002.

[9] A. Rogozhnikov, "Gradient boosting explained [demonstration]," 2016.

[10] L. Toscano, "Introduction to gradient-boosted trees and xgboost hyperparameters tuning (with python)," 2017.

[11] E. Cai, "Machine learning lesson of the day – overfitting and underfitting," 2014.

[12] R. Zacharski, *A Programmers Guide to Data Mining*. Creative Commons, 2015.

[13] W. Meira, Jr. and M. J. Zaki, *Data Mining and Analysis*. Cambridge University Press, 2014.