Gradient Tree Boosting for Housing Price Prediction

Steven Bogaerts[1], Utkarsh Singh[2], Sreya Chakrabarti[3]

Abstract

Accurate prediction of housing prices would be of significant benefit to consumers and the economy in general. Hedonic regression is one approach of use in this effort, in which an attempt is made to predict housing prices based on the weighted contributions of many attributes. This paper describes an application of this idea to the Ames, Iowa, USA dataset [1]. The dataset and several steps of data cleaning are presented, followed by a summary of two key regression algorithms. A variety of algorithms are tested, with a gradient tree boosting algorithm showing the best results. This model is then tuned further through a series of experiments, leading to a root mean squared log error of 0.13102 on a previously unseen testing set.

Keywords

gradient tree boosting — housing prices — hedonic regression — data cleaning

[1]**Corresponding author**: stevenbogaerts@depauw.edu, *Department of Computer Science, DePauw University, Greencastle, IN, USA*
[2]utksingh@umail.iu.edu, *Department of Data Science, School of Informatics and Computing, Indiana University, Bloomington, IN, USA*
[3]sreychak@umail.iu.edu, *Department of Data Science, School of Informatics and Computing, Indiana University, Bloomington, IN, USA*

Contents

Introduction 1
1 Background 1
2 Algorithm and Methodology 2
2.1 The Data . . . 2
2.2 Data Cleaning . . . 2
2.3 Algorithms . . . 4
Ordinary Least Squares Regression • Gradient Tree Boosting
3 Experiments and Results 6
3.1 Initial Exploratory Experiment . . . 6
3.2 Tuning GradientBoostingRegressor . . . 7
3.3 Future Work . . . 8
4 Summary and Conclusions 8
Acknowledgments 9
References 9

All the work herein is solely ours. (But built upon the work of many others as cited throughout.)

Introduction

With a home being the largest purchase most people will ever make, it is not surprising that housing price prediction is a frequently-studied task. Highly reliable price predictions would bring many economic benefits. Sellers and buyers could both be certain of a fair deal. Realtors could set prices perfectly according to the seller's goals, whether to sell quickly at a low price or more slowly at a somewhat higher price. Tax assessors could give accurate valuations of homes. And perhaps most intriguingly, civic organizations and government officials could run models of the effect of various policies on housing prices, enabling more effective building regulations, zoning laws, and city planning.

Of course there are many barriers to achieving this ideal. Many interacting components affect price, including not just physical features of the home, but buyer demographics, the rental market, land prices, construction costs, the regulatory environment, crime rates, school quality, and proximity to various facilities, to name a few. At present, the most accurate way to determine a home's value is to sell it. Of course this is not always a desired course of action, and may not even reflect worth accurately. For example, emotions of the buyer and seller can affect sale price, in the form of exhaustion with the buying/selling process, idiosyncratic preferences, or even personal feelings about others involved in the transaction.

This paper nevertheless explores strategies for making housing price predictions using the Ames, Iowa, USA dataset [1]. Further background on the problem is provided, followed by an explanation of the data cleaning steps and the application of gradient tree boosting, a series of experiments, and results.

1. Background

In this paper we explore the common method of *hedonic regression* in housing price prediction. In this approach, the price of a house is estimated based on a variety of attributes, each assumed to contribute to varying degrees to the true worth. The amount of contribution of each attribute to the price is determined via some form of regression analysis, with many options possible.

Many potential attributes seem rather obvious. For example, square footage should clearly be considered, because homes with greater square footage tend to sell for more. There are many additional attributes of this type representing physi-

cal qualities of the house, such as construction material, number of storeys, bedrooms, and bathrooms, year constructed, kitchen quality, etc. The Ames dataset considered here includes numerous such attributes as described in more detail below.

A number of researchers, however, have demonstrated the importance and complexity of many other attributes as well – attributes not directly connected to the physical house itself. For example, [2] and [3] study the influence of transportation and access to facilities. On the one hand, proximity to larger roads means greater potential for noise pollution and traffic, therefore reducing home prices. On the other hand, significant distance from such thoroughfares can mean reduced access to facilities, and therefore reduced home prices for the opposite reason. They show that simplified models of access based on a theoretical central hub are insufficient for accurate price analysis. Furthermore, access is not always desirable, as university and commerce access often leads to higher prices, while K-12 education and industry access often leads to lower prices.

Considering other forms of transportation, proximity to a subway stop can boost home prices in some markets [4], while proximity to an airport can either increase or decrease home prices depending on the presence of noise pollution [5]. In most transportation modes, the price boost of proximity can also be offset by a reduction in air quality, particularly in areas with higher average incomes [6].

Broad categories of locations can also greatly influence home prices. [7] demonstrates the different effects of physical improvements and neighborhood quality in suburban versus urban environments, leading to a recommendation for entirely different price models for the two categories. The number of land-use and building regulations can reduce vacant land values but increase home values [8], with such regulations being a major factor in high prices in San Francisco [9] and Manhattan [10], and in a study of 56 major cities across the United States [11]. Furthermore, not just the immediate regulatory environment but also the number of nearby distinct jurisdictions affect home prices in interacting ways [12].

Further potential attributes for hedonic regression include the rental market, land prices, and constructions costs [13], school quality [14], presence of registered sex offenders [15], and even the education level and age of typical local buyers [16]. The list of potentially relevant attributes in a hedonic regression seems nearly endless, such that even the impressively comprehensive Ames, IA dataset is inadequate for perfect prediction. Since one cannot simply wait for "better data," however, we nevertheless proceed in developing a model of housing prices on this dataset.

2. Algorithm and Methodology

2.1 The Data

As mentioned previously, the dataset considered in this paper is the Ames, Iowa, USA dataset [1]. At time of writing, this dataset is also the subject of a competition on kaggle.com [17]. The dataset includes 2,919 elements, each with a unique ID. Of these, 1,460 include a class variable, SalePrice, making up the training set. The testing set is made of the remaining 1,459 with an unknown SalePrice (for the purposes of the Kaggle competition). Thus all model construction is based on the 1,460-element training set.

Both the training and testing sets contain 79 additional attributes, primarily focused on the physical characteristics of the house itself. Nominal, ordinal, discrete, and continuous variables are included, with some elements of the dataset including missing values. The type of each attribute is specified below:

- **Nominal attributes (28)**: MSSubClass, MSZoning, Street, Alley, LotShape, LandContour, Utilities, LotConfig, LandSlope, Neighborhood, Condition1, Condition2, BldgType, HouseStyle, RoofStyle, RoofMatl, Exterior1st, Exterior2nd, MasVnrType, Foundation, Heating, CentralAir, Electrical, GarageType, Paved-Drive, MiscFeature, SaleType, SaleCondition.

- **Ordinal attributes (17)**: ExterQual, ExterCond, BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, HeatingQC, KitchenQual, Functional, FireplaceQu, GarageFinish, GarageQual, GarageCond, PoolQC, Fence, MoSold.

- **Discrete attributes (16)**: OverallQual, OverallCond, YearBuilt, YearRemodAdd, BsmtFullBath, BsmtHalfBath, FullBath, HalfBath, Bedroom, Kitchen, TotRmsAbvGrd, Fireplaces, GarageYrBlt, GarageCars, GarageArea, YrSold.

- **Continuous attributes (18)**: LotFrontage, Lot Area, MasVnrArea, BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, 1stFlrSF, 2ndFltSF, LoqQualFinSF, GrLivArea, WoodDeckSF, OpenPorchSF, EnclosedPorch, 3SsnPorch, ScreenPorch, PoolArea, MiscVal.

In judging which attribute falls into which type above, for attributes reflecting "quality" (e.g. ExterQual, BsmtQual, OverallQual), we considered attributes with *Excellent* / *Good* / ... values to be ordinal, since differences between successive ratings may not have consistent meaning. In contrast, we judged "quality" attributes with numerical values (e.g. scale 1-10) as discrete. For a complete description of all attributes and possible values, see [17].

2.2 Data Cleaning

Perhaps the greatest proportion of total effort in this work was spent on preparing the data. We addressed missing attributes in a variety of ways, followed by a series of numeric transformations and conversions, as described below. This and all programming work was done using Python 3.5.2 with SciPy 0.18.1, NumPy 1.11.2, Pandas 0.19.0, and scikit-learn 0.18.

In each step of data cleaning, we adhered to a principal of using only the training set for calculations, in the hopes of obtaining better generalizability. So, while we do transform both the training and testing sets in a consistent manner, the transformations are based only on calculations from the training set.

The dataset uses the string *NA* to indicate a missing value. Sometimes *NA* truly means the value is unknown, while at other times it seems to suggest a valid "not applicable" value for the attribute. Thus, each attribute with missing values was handled based on our intuition about how best to maintain the most appropriate semantics. Specifically:

- *Alley, Utilities, BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, FireplaceQu, GarageType, GarageFinish, GarageQual, GarageCond, PoolQC, Fence, MiscFeature*: For these attributes, it is meaningful to interpret *NA* as "not applicable" as opposed to "missing." For example, Alley can be *Gravel*, *Paved*, or *NA*, where *NA* means "No alley access". Thus *NA* here does not indicate a missing value, but rather that the attribute is not applicable. *NA* for Utilities is taken to mean that the house has no connected utilities. *NA* for any basement-related attribute (prefixed with "Bsmt") indicates that there is no basement; similarly for fireplace, garage, pool, fence, and any miscellaneous feature. Thus for each of these attributes, we treat *NA* as a valid nominal or ordinal value, not as a missing value. Thus no replacement is performed for these attributes.
- *MSZoning, Exterior1st, Exterior2nd, MasVnrType, Electrical, KitchenQual, Functional, SaleType*: In contrast, these attributes must have a value; "not applicable" does not make sense, and so *NA* means "missing" and must be addressed. Specifically:
  - MSZoning must have a value because real estate must be zoned. Thus *NA* indicates a missing value.
  - Exterior1st must have a value, because a home must have an exterior. Thus *NA* indicates a missing value. Only a single house in the dataset is missing Exterior2nd, so we treat it as Exterior1st.
  - MasVnrType can already have the value *None*, so *NA* for "not applicable" does not make sense. Thus *NA* indicates a missing value.
  - Electrical has *NA* only for a single house built in 2007. Since it is unlikely that this house was built without access to electricity, we consider this a missing value.
  - KitchenQual has a missing value only for a single house, but this house has a value of 1 for the number of kitchens. Thus the KitchenQual value is missing.
  - Functional describes the home functionality, ranging from "typical functionality" to "salvage only." Each home must fall into one of these categories, and so *NA* indicates a missing value.
  - SaleType describes the type of sale for the home. Each home in the dataset has been sold, and so *NA* indicates a missing value.

  Thus for each of the above nominal and ordinal attributes, *NA* is replaced with the mode for the attribute in the training set.
- *MasVnrArea, BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, BsmtFullBath, BsmtHalfBath, GarageCars, GarageArea*: For each of these numerical attributes (either discrete or continuous), some data elements have the value 0. This suggests that *NA* is not intended to mean 0 here, and so *NA* is considered to indicate a missing value and replaced with the mean value among the training set.
- *LotFrontage*: This represents the number of linear feet of street connected to the property. Not a single data element contains a value of 0 for this attribute, so this suggests possibly that *NA* really means 0 here. However, across the dataset there are 486 elements (16.6%) with the value *NA* for this attribute. It seems unlikely that 16.6% of homes in Ames, IA have no adjacent streets, and so we interpret *NA* to indicate a missing value, and replace with the mean LotFrontage value among the training set.
- *GarageYrBlt*: *NA* means that there is no garage. One option would be to use this as a valid value and consider GarageYrBlt to be a nominal attribute. However, this would mean the loss of some information, as the use of a numerical year would then be obscured in considering this a nominal attribute. Note that the absence of the garage is already accounted for in a GarageCars attribute value of 0. So, in order to maintain the type of GarageYrBlt as a numerical year, we replace each data element's *NA* for GarageYrBlt with the year the corresponding house itself was built.

The remaining attributes have no missing values, and so the above addresses all missing values.

Next, our code proceeds to a series of numeric transformations and conversions:

- *Years Since*: In order to bring the year-based attributes (YearBuilt, YearRemodAdd, GarageYrBlt, and YrSold) into a lower numerical range, we replace each year y with $2016 - y$, with 2016 being the year in which this paper was written. Thus, each year-based attribute is now a measure of "years since."
- *Log Transformation for Skewed Attributes*: A number of numeric attributes are highly skewed, with only a few outliers containing higher values. To address

Table 1. The average sale price in US$ for each neighborhood, in the training set.

Nbrhd.	Avg. Price	Nbrhd.	Avg. Price
MeadowV	$98,576.47	NWAmes	$189,050.07
IDOTRR	$100,123.78	Gilbert	$192,854.51
BrDale	$104,493.75	Blmngtn	$194,870.88
BrkSide	$124,834.05	CollgCr	$197,965.77
Edwards	$128,219.70	Crawfor	$210,624.73
OldTown	$128,225.30	ClearCr	$212,565.43
Sawyer	$136,793.14	Somerst	$225,379.84
Blueste	$137,500.00	Veenker	$238,772.73
SWISU	$142,591.36	Timber	$242,247.45
NPkVill	$142,694.44	StoneBr	$310,499.00
NAmes	$145,847.08	NridgHt	$316,270.62
Mitchel	$156,270.12	NoRidge	$335,295.32
SawyerW	$186,555.80		

this, we first determine the skewness of each numerical attribute using the Fisher-Pearson standardized third moment coefficient as implemented in SciPy and demonstrated in [18]. By this test, the following numerical attributes were the most highly-skewed: MsSubClass, LotFrontage, LotArea, MasVnrArea, BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, 1stFlrSF, 2ndFlrSF, LowQualFinSF, GrLivArea, BsmtHalfBath, KitchenAbvHr, WoodDeckSF, OpenPorchSF, EnclosedPorch, 3SsnPorch, ScreenPorch, PoolArea, and MiscVal. For each, we applied the transformation $t_1(x) = \log(1+x)$ to each attribute value x, in an attempt to bring these attributes closer to a normal distribution.

- *Normalization of Numeric Attributes*: All numeric attributes are normalized to the range $[0,1]$ by dividing by the maximum observed value in the training set. Thus, unless the value 0 actually occurs for a given attribute in the original data, 0 will not occur in the normalized attribute values either. It is also possible that a high outlier in the testing set could result in a value above 1, but we chose not to worry about this, making all transformations be based only on training set data.

- *Normalization of the Neighborhood Attribute*: We expect that the nominal attribute Neighborhood is a particularly important attribute in predicting house price, and so special attention was given here. The average price for each neighborhood was calculated and ordered. An ordered list of average price per neighborhood is presented in Table 1. Each Neighborhood attribute value x was then converted to a single numerical value $t_2(x)$ in the range $[0,1]$ as follows:

$$t_2(x) = \frac{\sum_{i \in N_x} p_i}{\sum_{j \in N} p_j}$$

where N is the set of all neighborhoods, p_i is the average price of a neighborhood $i \in N$, and N_x for some neighborhood x is the set of neighborhoods such that $\forall i \in N_x, p_i \leq p_x$. By this transformation, for example, the lowest-price *MeadowV* neighborhood becomes 0.214, *Sawyer* becomes 0.179, and the highest-price *NoRidge* becomes 1. In this way, we convert the nominal attribute Neighborhood into a continuous one while not imposing an arbitrary ordering on the values; rather, the ordering is based on average price.

- *Conversion to Dummy Attributes*: All other non-numeric attributes are converted into multiple dummy attributes. This process is sometimes called one hot encoding. For example, consider the attribute MasVnrType, which describes the type of masonry on the home. Possible values for this nominal attribute are *BrkCmn*, *BrkFace*, *CBlock*, *None*, and *Stone*. To convert MasVnrType to multiple dummy attributes, we replace the MasVnrType attribute with binary attributes MasVnrType_BrkCmn, MasVnrType_BrkFace, MasVnrType_CBlock, MasVnrType_None, and MasVnrType_Stone. For each element of the dataset, exactly one of these binary attributes will take on the value 1 (corresponding to the value of MasVnrType) while the others will be 0. In this way non-numeric attributes can be converted into numeric attributes without introducing an arbitrary ordering of values.

The above steps produce a set of entirely numeric and normalized attributes on which regression analysis can be conducted.

2.3 Algorithms

As will be shown, initial experiments suggest that a gradient boosting regressor is the most effective algorithm among those we attempted on this dataset, thus we discuss the algorithm here. We begin with an introduction to regression, followed by an application of gradient boosting to regression.

2.3.1 Ordinary Least Squares Regression

Regression is an effort to find a relationship between a set of independent variables and a single dependent variable. One simple approach is *linear regression*, in which we attempt to approximate the dependent variable y with a linear equation $h_{\vec{\theta}}(\vec{x}) = \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_m x_m + \theta_0$, for vector $\vec{x}$ consisting of independent variables x_1 through x_m and vector $\vec{\theta}$ consisting of constants θ_0 through θ_m. Sometimes this is called *multiple regression* because there are multiple independent variables. $h_{\vec{\theta}}$ is called the *linear regression model* or simply the *linear model*, with *parameters* θ_0 through θ_m. This is called a model because it is unlikely that $h_{\vec{\theta}}$ perfectly computes y for each element of the dataset.

Regression analysis is the application of some algorithm to find the parameters θ_0 through θ_m that best match the available data. The precise definition of "best match" is often given by whichever parameters minimize a *loss function*. Many

possible loss functions exist, but one common one is the sum of the squares of the differences between the dataset's dependent variable values and those predicted by the model. More precisely, this loss function is defined:

$$J(\vec{\theta}) = \frac{1}{2}\sum_i [h_{\vec{\theta}}(\vec{x}^{(i)}) - y^{(i)}]^2$$

where $\vec{x}^{(i)}$ and $y^{(i)}$ are the values of $\vec{x}$ and y for the ith data element. Linear regression using $J(\vec{\theta})$ is called *ordinary least squares*. Whether the minimizing values of $\vec{\theta}$ gives a "reasonable" approximation is open to the discretion of the practitioner, with a common approach being to use some threshold maximum for $J(\vec{\theta})$.

In many cases, this minimization problem has a closed-form solution. Define

$$\vec{X} = \begin{bmatrix} (\vec{x}^{(1)})^T \\ (\vec{x}^{(2)})^T \\ \vdots \\ (\vec{x}^{(n)})^T \end{bmatrix}$$

and

$$\vec{Y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

where n is the number of elements in the dataset. To minimize $J(\vec{\theta})$, compute:

$$\vec{\theta} = (\vec{X}^T\vec{X})^{-1}(\vec{X}^T\vec{Y})$$

Occasionally, however, this may not be appropriate, because the inverse $(\vec{X}^T\vec{X})^{-1}$ does not exist or is deemed too costly to compute. In that case, $J(\vec{\theta})$ can be minimized numerically, by visualizing $J(\vec{\theta})$ as a surface in $(m+1)$-dimensional space for the model parameters θ_0 through θ_m, and performing gradient descent. That is, iteratively update

$$\theta_j \leftarrow \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\vec{\theta})$$

for $0 \le j \le m$ and some constant *learning rate* α. Since $\frac{\partial}{\partial\theta_j}J(\vec{\theta}) = (h_{\vec{\theta}}(\vec{x}) - y)x_j$, this is equivalent to

$$\theta_j \leftarrow \theta_j - \alpha x_j(h_{\vec{\theta}}(\vec{x}) - y)$$

Note that for θ_0 we can suppose $x_0 = 1$. Figure 1 provides pseudocode for a batch-update iterative ordinary least squares process.

Ordinary least squares is just one of many regression techniques, each defined using alternative loss functions and regression models. For other regression techniques where a closed form is not readily available, the gradient descent approach (modified for other loss functions) can be used.

Ordinary Least Squares
INPUT $(X = \{\vec{x}^{(i)}\}, Y = \{y^{(i)}\}, \alpha)$
▷ There are n data elements, each with m independent variable values. That is, $i \in \mathbb{Z}$, $1 \le i \le n$, and $\forall i, \vec{x}^{(i)}$ has m elements.
OUTPUT $\vec{\theta}$
$\vec{\theta} \leftarrow$ arbitrary initial values
repeat
  for $i = 1, n$ **do** ▷ For each data element $(\vec{x}^{(i)}, y^{(i)})$
    $x_0^{(i)} \leftarrow 1$ ▷ To update θ_0
    for $j = 0, m$ **do** ▷ For each model parameter θ_j
      $t_j \leftarrow \theta_j - \alpha x_j^{(i)}(h_{\vec{\theta}}(\vec{x}^{(i)}) - y^{(i)})$
    end for
    $\vec{\theta} \leftarrow \vec{t}$ ▷ Batch update
  end for
until change in $\vec{\theta}$ is below some threshold
return $\vec{\theta}$

Figure 1. Pseudocode for a batch-update iterative ordinary least squares process.

2.3.2 Gradient Tree Boosting

In machine learning, an *ensemble* is a collection of predictors that together provide an aggregate prediction for some problem by combining their individual predictions. One simple type of ensemble is an *additive* model in which the final prediction is a weighted sum of the individual predictions:

$$h(\vec{x}) = \sum_t \rho_t h_t(\vec{x})$$

where h_t for some t denotes a single predictor with weight ρ_t. In the simplest case, as we will consider here, $\forall t, \rho_t = 1$, and so $h(\vec{x}) = \sum_t h_t(\vec{x})$.

Many strategies for developing the components of an ensemble exist. In regression, we can define a *strong learner* as a regressor with low error (by an arbitrary definition of "low"), while a *weak learner* is a regressor with higher error. *Boosting* is one strategy of ensemble development, in which weak learners are iteratively added to an ensemble, with the intention of creating a strong learner in the aggregate [19]. *Gradient tree boosting*, aka *gradient boosted regression trees*, is a particular boosting strategy [20].

A *regression tree* is a form of decision tree used to predict a continuous value, as opposed to the classification prediction of ordinary decision trees. In their simplest form, regression trees predict some constant value at each leaf. For example, it may predict the mean of the dependent variable across the subset of the training set that lies at that leaf. A more complex approach would be to apply, for example, a local linear regression at each leaf.

In gradient tree boosting, the weak learners making up the ensemble are regression trees. In the spirit of boosting, each regression tree may not be particularly effective on its own, but trees are iteratively added to the ensemble in order to address portions of the problem space with highest prediction

Gradient Tree Boosting
INPUT $(X = \{\vec{x}^{(i)}\}, Y = \{y^{(i)}\}, \alpha)$
▷ There are n data elements, and so $i \in \mathbb{Z}$, $1 \leq i \leq n$.
OUTPUT $h^{(t)}$
$t \leftarrow 0$
$h^{(t)} = \sum_i y^{(i)}/n$
repeat
$h_{t+1} \leftarrow RegressionTree(\{(\vec{x}^{(i)}, y^{(i)} - h^{(t)}(\vec{x}^{(i)}))\})$
$h^{(t+1)} \leftarrow h^{(t)} + h_{t+1}$
$t \leftarrow t+1$
until convergence
return $h^{(t)}$

Figure 2. Pseudocode for the gradient tree boosting algorithm.

error according to some loss function.

The following algorithm discussion is based on [21]. Suppose we have some initial predictor $h^{(0)}(\vec{x})$. This can be any simple approach like $h^{(0)}(\vec{x}) = \sum_i y^{(i)}/n$. This model likely contains a number of prediction errors, and so we wish to add a new regression tree $h_1(\vec{x})$ to obtain $h^{(1)}(\vec{x}) = h^{(0)}(\vec{x}) + h_1(\vec{x})$. Again, we are here assuming an additive ensemble with $\forall t, \rho_t = 1$. More generally, this iterative process can be summarized with:

$$h^{(0)}(\vec{x}) = \sum_i y^{(i)}/n$$
$$h^{(t+1)}(\vec{x}) = h^{(t)}(\vec{x}) + h_{t+1}(\vec{x}) = \sum_t h_t(\vec{x})$$

where each $h^{(t)}$ for $t > 0$ is a regression tree.

The question remains, then, how to define h_{t+1} at each step in the iteration. Of course we hope that $h^{(t)}(\vec{x}) + h_{t+1}(\vec{x})$ will give a perfect prediction. That is, we hope that:

$$\forall i, h^{(t)}(\vec{x}^{(i)}) + h_{t+1}(\vec{x}^{(i)}) = y^{(i)}$$

Rearranging the terms, we can define as our goal:

$$\forall i, h_{t+1}(\vec{x}^{(i)}) = y^{(i)} - h^{(t)}(\vec{x}^{(i)})$$

That is, we want to define a regression tree h_{t+1} over the examples $\forall i, (\vec{x}^{(i)}, y^{(i)} - h^{(t)}(\vec{x}^{(i)}))$ as closely as possible. These $y^{(i)} - h^{(t)}(\vec{x}^{(i)})$ terms are called *residuals*. It can be shown that this regression problem is equivalent to gradient descent on a loss function, hence the name gradient tree boosting. Figure 2 summarizes the algorithm based on the discussion above. While the pseudocode specifies to repeat until "convergence", many possible termination criteria exist, including observing when the change in error on the training set is below some threshold.

3. Experiments and Results

All experiments were executed on a dual-socket, 12-core Intel Xeon system with 512GB of memory, available at Indiana University - Bloomington. As mentioned above, we used Python 3.5.2 with SciPy 0.18.1, NumPy 1.11.2, Pandas 0.19.0, and scikit-learn 0.18 for all coding in this work.

Recall that the complete Ames, IA dataset is split into a training set, for which housing prices are known, and a testing set, for which housing prices are not known (except by the Kaggle contest organizers). This testing set is used only in making a submission to the Kaggle contest site, and therefore is not considered here until the final experiment. In our experiments, we use k-fold cross-validation on the Ames, IA training set. In this process, the Ames, IA training set is split into an *experimental training set* and *experimental test set*.

Unless specified otherwise, algorithm performance was measured using R^2, reporting the average R^2 value across each of the k folds. In the R^2 measure, 1.0 is the best-possible score, meaning that the predicted house prices matched the known training set house prices perfectly in every example. 0.0 represents the score of a constant model that always predicts the mean of the house prices in the training set – thus 0.0 is a baseline score of a trivial implementation. Negative scores are also possible, indicating a model that performs worse than the constant model.

All experiments were performed using the entire set of attributes for prediction, except of course the unique identifier and the *SalePrice* variable to be predicted. Attributes were modified as described in the Data Cleaning section above.

3.1 Initial Exploratory Experiment

Through scikit-learn, it is relatively easy to instantiate a variety of algorithm implementations and explore roughly how they perform on the dataset. Thus our first experiment considers a wide range of algorithms, with the intent to quickly narrow the field to a couple candidate algorithms to be explored more carefully. Each algorithm was evaluated with a single run of 3-fold cross-validation, using the default algorithm parameter values of the scikit-learn implementations.

Results are shown in Figure 3, with names corresponding to the scikit-learn class name. First in the figure is ElasticNet, with a comparatively poor 0.66 score. Through further tuning of the `alpha` parameter, an `alpha` value of 0.1 gave a score of 0.79 – a significant improvement, but not enough to warrant more careful consideration at this time compared to the other algorithms.

Figure 3 shows a number of algorithms achieving R^2 scores in the low 0.80s. The clear single stand-out in this first experiment, though, is the GradientBoostingRegressor at 0.8978. Thus the next experiment will explore this algorithm in more detail.

Not shown in Figure 3 are two surprising outliers: LinearRegression, and BaggingRegressor built upon LinearRegression. In both cases, an R^2 score of around $-1e20$ was obtained. Clearly, these results reflect some error. Upon examining the predicted house prices by LinearRegression, we found most in a reasonable range of around eighty thousand up to a few hun-

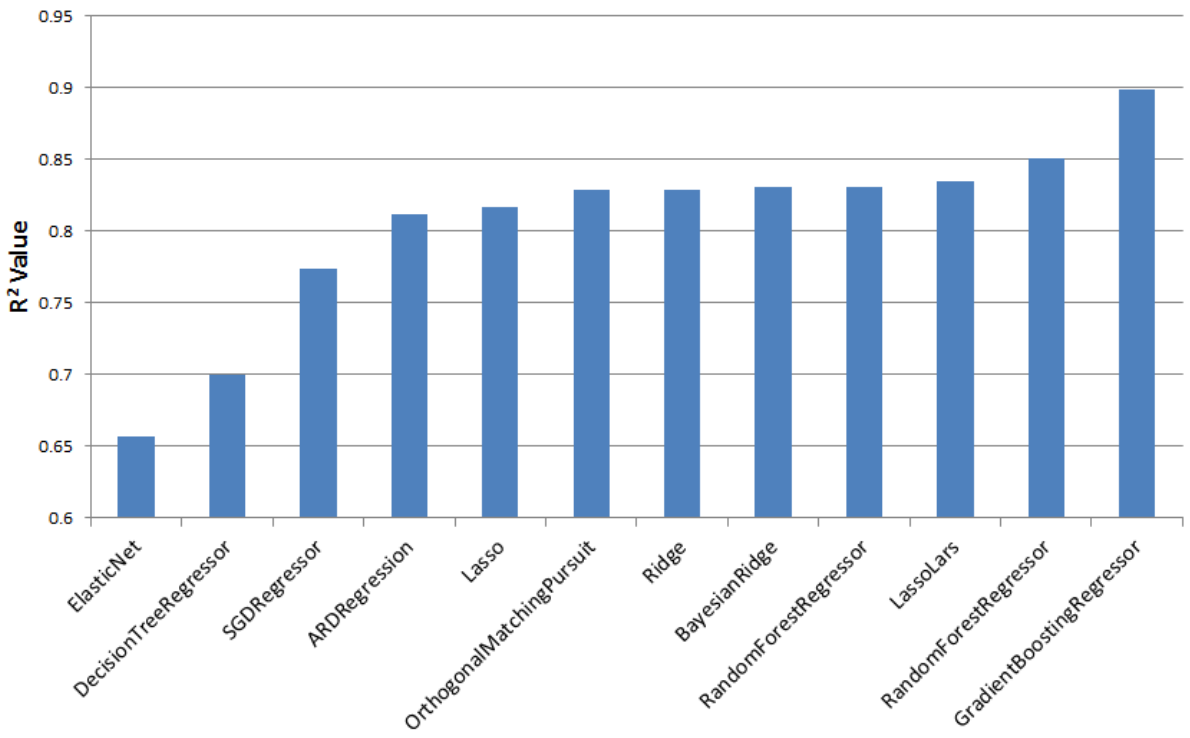

Figure 3. Results of the initial exploratory experiment, using all default parameter values in standard scikit-learn implementations.

dred thousand dollars. However, there are also a few dozen outliers with various values in the negative trillions of dollars. This is quite peculiar since a modification of the data cleaning steps removes this problem. For example, if instead of using dummy attributes, we convert each non-numeric attribute to a single numeric one in the range $[0,1]$ ordered by average price (as done with the Neighborhood attribute), the R^2 score for LinearRegression is around 0.75. Given this inferior result even with an alternative data cleaning strategy, we chose not to explore this behavior of LinearRegression further at this time.

3.2 Tuning GradientBoostingRegressor

Having determined that the GradientBoostingRegressor seems to be the best approach, we now consider tuning of the algorithm for improved results. [22] describes a tuning strategy for a GradientBoostingClassifier (not a regressor) in which initial values of various parameters are set and then tested one or two at a time. The order of testing of attributes is generally in order of their importance, with highly-related attributes tested simultaneously. When values tested in [22] were based on a proportion of the total number of samples, we used the same proportions to obtain our own values. In order to obtain consistency in this stochastic testing across multiple steps, we used $random\_state = 10$ throughout, to always work with the same random seed.

The parameters with which we experimented are described below, along with the scikit-learn default values:

- $learning\_rate$: The contribution of each tree to the ensemble. Default = 0.1.
- $min\_samples\_split$: The minimum number of samples from the dataset that are needed to split a node in the tree. Default = 2.
- $min\_samples\_leaf$: The minimum number of samples allowed in a leaf. Default = 1.
- max_depth: The maximum depth of each tree. Default = 3.
- $max\_features$: The maximum number of features to be considered in defining a single split criterion for a node. Default = all features.
- $subsample$: The proportion of samples to be randomly chosen to create each tree. Default = 1.0.
- $n\_estimators$: The number of trees to create. Default = 100.

The above default values gave the results of the experiment above for a GradientBoostingRegressor, with an R^2 score of 0.8978.

To begin this series of experiments, we set the parameters as $(learning\_rate = 0.1, min\_samples\_split = 11, min\_samples\_leaf = 2, max\_depth = 8, max\_features = sqrt, subsample = 0.8)$, with $n\_estimators$ to be varied first. Note that the value $max\_features = sqrt$ indicates that up to the square root of the total number of attributes should be used for each split criterion. These are just initial values, to be tuned later. As already mentioned, $min\_samples\_split$,

min_samples_leaf, and *max_depth* are based on the same value-to-number-of-samples ratio used in the example dataset of [22], while all other values match that work precisely.

We then considered *n_estimators* values in the set $\{20,40,60,80,100,120,140,160,180,200,250\}$. Again, this range was determined using proportionally the same values as in [22]. Results showed that $n\_estimators = 120$ was ideal, with an R^2 value of 0.8835. Note that this is lower than the R^2 score with default values obtained above, but this will be improved upon below.

With *n_estimators* set at 120, we turned our attention to *max_depth* and *min_samples_split* simultaneously. Using a cross-validation grid search with 5 folds, we explored *max_depth* values in $\{1,2,3,4,5,6,7,8,10,12,14,16,18,20\}$ and *min_samples_split* values in $\{2,4,6,8,10,12,14,16,18,20,22,24,26,28,30\}$. Results showed that $max\_depth = 6$ and $min\_samples\_split = 12$ gave the best R^2 score, of 0.89628, and so we fixed those parameters accordingly.

Next we simultaneously explored *min_samples_leaf* and *min_samples_split* values of $\{1,2,3,4,6,8,10,15,20\}$ and $\{2,4,6,8,10,12,14,16,18,20,22,24,26,28,30\}$, respectively. Using the same grid search as above, the best combination of $min\_samples\_leaf = 1$ and $min\_samples\_split = 24$ gave an R^2 score of 0.8992.

The next experiment considered values of *max_features* in $\{2,4,6,8,10,12,14,16,18,20,22,24\}$, with results finding $max\_features = 16$ giving the best R^2 score, of 0.8992.

The next experiment considered values of *subsample* in $\{0.6,0.7,0.75,0.8,0.85,0.9\}$, with results finding $subsample = 0.8$ giving the best R^2 score, still at 0.8992.

Finally, we returned to probably the most important parameters again: *n_estimators* and *learning_rate*. Testing values of *learning_rate* in $\{0.01,0.02,0.04,0.06,0.08,0.10,0.12,0.14,0.16,0.18,0.20\}$ and *n_estimators* in $\{20,40,60,80,100,120,140,160,180,200,250,300,350,400,500,600,700\}$, the best combination of $n\_estimators = 400$, $learning\_rate = 0.06$ gave an R^2 score of 0.9014. This is an improvement of 0.0036 over the score with the default parameter values.

To summarize, the above experiments suggest the following tuned parameter settings:

- $learning\_rate = 0.06$
- $min\_samples\_split = 12$
- $min\_samples\_leaf = 1$
- $max\_depth = 6$
- $max\_features = 16$
- $subsample = 0.8$
- $n\_estimators = 400$

When a GradientBoostingRegressor with these parameter values is applied to the testing set from the Kaggle competition, a root mean squared log error (RMSLE) score of 0.13102 is obtained. At time of writing, this corresponds to 1096^{th} place out of 2545 competitors. By way of comparison, at time of writing the top three teams in this Kaggle competition have RMSLE scores of 0.07186, 0.09403, and 0.10051, respectively.

3.3 Future Work

Much work remains that could improve these results further. In data cleaning, for example, the use of dummy variables could be limited to just nominal types, while ordinal types could be converted to a single value in $[0,1]$. This might be acceptable for ordinal types since they are already ordered, but such a conversion would impose a distance between ordered items not implied in the original data. The possible benefit of this approach, though, is that fewer attributes would exist than in using dummy variables for all non-numeric data, and thus a regressor might learn more easily. It is also possible that the use of fewer dummy variables could make principal component analysis (PCA) more effective, as the number of attributes to consider in PCA from the beginning could be greatly reduced.

Another data cleaning idea is to introduce new attributes. One key attribute that may play an important role would be to convert each target value from a total dollar amount to dollars per square foot. A regressor could be trained on this modified target attribute, with the regressor's results then converted back to a total dollar amount for a final prediction. This approach may increase the number of homes that appear comparable in price when considering attributes other than square footage, providing, in effect, more examples per price point upon which to build a model.

This work focused on using the complete set of attributes as predictors. It is possible that a subset, either chosen by hand or obtained algorithmically, could provide better results.

The experiments described above are just the beginning of what could be a more comprehensive study. The erroneous linear regression results, of course, need to be explored. A wider variety of algorithms, with some tuning of each, over more trials, could lead to a more effective model as well. The most promising algorithms could be tuned even further than the process described above, using more potential values or even an optimization algorithm on the parameter values.

4. Summary and Conclusions

This paper presents the challenges of predicting housing prices. Several data cleaning steps are described, along with summaries of ordinary least squares regression and gradient tree boosting. An exploratory experiment showed that among several algorithms, gradient tree boosting is most effective. The algorithm parameters are then tuned to achieve improved results.

Acknowledgments

We would like to thank Dr. Mehmet Dalkilic and the assistants led by Hasan Kurban for their work in Data Mining B565 Fall 2016 at Indiana University. We also thank the authors of the

materials cited throughout this paper for their very helpful previous work and tutorials.

References

[1] Dean De Cock. Ames, iowa: Alternative to the boston housing data as an end of semester regression project. *Journal of Statistics Education*, 19(3), 2011.

[2] Joel P Franklin and Paul Waddell. A hedonic regression of home prices in king county, washington, using activity-specific accessibility measures. In *Proceedings of the Transportation Research Board 82nd Annual Meeting, Washington, DC*, 2003.

[3] Mingche M Li and H James Brown. Micro-neighborhood externalities and hedonic housing prices. *Land economics*, 56(2):125–141, 1980.

[4] Vladimir Bajic. The effects of a new subway line on housing prices in metropolitan toronto. *Urban Studies*, 20(2):147–158, 1983.

[5] Jeffrey P Cohen and Cletus C Coughlin. Spatial hedonic models of airport noise, proximity, and housing prices. *Journal of Regional Science*, 48(5):859–878, 2008.

[6] David Harrison and Daniel L Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management*, 5(1):81–102, 1978.

[7] Allen C Goodman. Hedonic prices, price indices and housing markets. *Journal of Urban Economics*, 5(4):471–484, 1978.

[8] Henry O Pollakowski and Susan M Wachter. The effects of land-use constraints on housing prices. *Land Economics*, 66(3):315–324, 1990.

[9] Lawrence Katz and Kenneth T Rosen. The interjurisdictional effects of growth controls on housing prices. *The Journal of Law & Economics*, 30(1):149–160, 1987.

[10] Edward L Glaeser, Joseph Gyourko, and Raven Saks. Why is manhattan so expensive? regulation and the rise in housing prices. *Journal of Law and Economics*, 48(2):331–369, 2005.

[11] Stephen Malpezzi. Housing prices, externalities, and regulation in us metropolitan areas. *Journal of Housing Research*, 7(2):209, 1996.

[12] Keith R Ihlanfeldt. The effect of land use regulation on housing and land prices. *Journal of Urban Economics*, 61(3):420–435, 2007.

[13] Michael J Potepan. Explaining intermetropolitan variation in housing prices, rents and land prices. *Real Estate Economics*, 24(2):219–245, 1996.

[14] Thomas J Kane, Stephanie K Riegg, and Douglas O Staiger. School quality, neighborhoods, and housing prices. *American Law and Economics Review*, 8(2):183–212, 2006.

[15] Jaren C Pope. Fear of crime and housing prices: Household reactions to sex offender registries. *Journal of Urban Economics*, 64(3):601–614, 2008.

[16] Richard Green and Patric H Hendershott. Age, housing demand, and real house prices. *Regional Science and Urban Economics*, 26(5):465–480, 1996.

[17] House prices: Advanced regression techniques. https://www.kaggle.com/c/house-prices-advanced-regression-techniques. Accessed: 2016-12-05.

[18] Alexandru Papiu. Regularized linear models. https://www.kaggle.com/apapiu/house-prices-advanced-regression-techniques/regularized-linear-models. Accessed: 2016-12-05.

[19] Leo Breiman et al. Arcing classifier (with discussion and a rejoinder by the author). *The annals of statistics*, 26(3):801–849, 1998.

[20] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[21] Cheng Li. A gentle introduction to gradient boosing. http://www.chengli.io/tutorials/gradient_boosting.pdf. Accessed: 2016-12-12.

[22] Aarshay Jain. Complete guide to parameter tuning in gradient boosting in python. https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/. Accessed: 2016-12-12.