# House Price Estimation Using Gradient Boosting, Random Forest and Linear Regression Methods

Gabrielle Belok
Boston University
Boston, MA
gabelok@bu.edu

Mehrnoosh Sarmashghi
Boston University
Boston, MA
msmshgi@bu.edu

Artin Spiridonoff
Boston University
Boston, MA
artin@bu.edu

## Abstract

*In this project we develop and implement different regression methods on datasets with a large number of both numerical and categorical features. We will explore linear regression as well as ensemble methods such as Gradient Boosting and Random Forest. We then evaluate each methods advantages along with their limitations such as overfitting.*

## 1. Introduction

The purpose of this project is to explore various regression methods on datasets with mixed features. Therefore we initially chose a relatively small dataset to develop and implement our methods. Then we applied our algorithms to a bigger dataset. In both datasets, we estimated the sale price of a house based on its various features such as it's area, garage, neighborhood, etc. The first dataset has a small number of datapoints, however it includes 80 categorical and numerical features, which still makes the house price estimation a challenging job. The second one has about 21 thousand datapoints with 21 features. We will use various linear regression methods, random forest and finally gradient boosting to predict the house price. We will then compare the performance of these methods.

The rest of this report is organized as following: In section 2 we briefly introduce and study our regression methods. In section 3 we describe the main dataset and our preprocessing method. In section 4 we describe the implemented algorithms and their results. Finally, we'll compare the results and conclude in section 5.

## 2. Solution Approaches

In this section, we describe the methods and models that we will study and implement in our project.

### 2.1. Linear Regression

[1] [2] Linear Regression is a supervised linear model in which the dependent variable $y$ is estimated using a linear function of independent variables $\mathbf{x} = (x_1, x_2, \ldots, x_d)^T$, *i.e.*,

$$\hat{y} = \hat{\beta} + \hat{\mathbf{w}}^T \mathbf{x} \tag{1}$$

Standard linear regression model makes a couple of assumptions about the dependent variable, independent variables and their relationship that we will explain in following:

**Linearity**: The conditional mean of the dependent variable is an affine function of the independent variables, *i.e.*:

$$E[Y|x_1, x_2, \ldots, x_d] = \beta + \mathbf{w}^T \mathbf{x} \tag{2}$$

or equivalently,

$$y_j = \beta + \mathbf{w}^T \mathbf{x}_j + \epsilon_j \tag{3}$$

where the error term $\epsilon_j$ is a zero mean random variable that adds noise to the linear model. $\beta$ is called the offset.

**Homoscedasticity:** Different values of the measured variable have the same variance in their errors, regardless of the values of the independent variables.

**Independence of errors:** The errors $\epsilon_j$'s are uncorrelated with each other.

Different approaches can be used to learn the linear regression model; The most common models are Ordinary Least Square (OLS), Ridge regression, Least Absolute Selection and Shrinkage Operation (LASSO) and Robust Regression.
Ordinary Least Square minimizes the sum of squared errors and has a closed form expression for the estimated model parameters, *i.e.*,

$$[\hat{\beta}, \hat{\mathbf{w}}^T]^T = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{y} \tag{4}$$

where $\mathbf{X} = (\mathbf{1}_d, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)^T$.

Ridge and LASSO differ from OLS in that they have an extra regularization term in their cost function. More specifically, they penalize coefficients by adding a $l_2$ and $l_1$ norm of the $w$ to the cost function, respectively. Regularization can improve the performance by not letting the coefficients become too large. More specifically, in Ridge Regression we have,

$$(\hat{\beta}, \hat{\mathbf{w}}) = \arg\min \left[ \sum_{j=1}^{n} (y_j - w^T \mathbf{x}_j - \beta)^2 + \alpha \|w\|_2^2 \right], \quad (5)$$

where $\alpha$ is the regularization strength. Similarly, in LASSO the cost function is,

$$(\hat{\beta}, \hat{\mathbf{w}}) = \arg\min \left[ \sum_{j=1}^{n} (y_j - w^T \mathbf{x}_j - \beta)^2 + \alpha \|w\|_1 \right]. \quad (6)$$

LASSO can improve the performance by setting the coefficients of irrelevant features to zero, while they might have values close to zero using ridge.

Finally, in robust regression, the squared error in the cost function is replaced with a special function of error which is less sensitive to outliers, *i.e*.,

$$(\hat{\beta}, \hat{\mathbf{w}}) = \arg\min \left[ \sum_{j=1}^{n} \rho(y_j - w^T \mathbf{x}_j - \beta) + \alpha \|w\|_2^2 \right]. \quad (7)$$

Different types of Robust Regression can be obtained based on the choice of $\rho$. In this project we used Huber robust regression in which, the $\rho$ is defined as follows:

$$\rho(x) = \begin{cases} \frac{1}{2}x^2 & \text{if } x < \delta, \\ \delta(|x| - \frac{1}{2}\delta) & \text{otherwise.} \end{cases} \quad (8)$$

where the value $\delta$ is chosen based on the distribution and variance of the data.

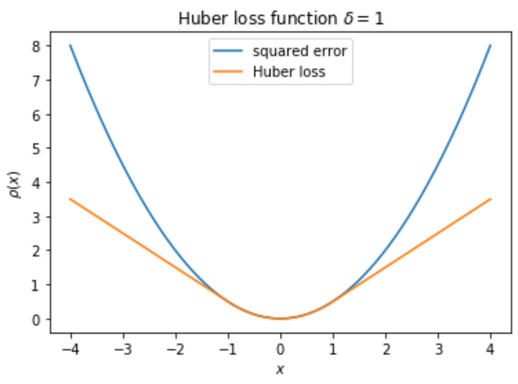

Figure 1. Huber loss function for $\delta = 1$

## 2.2. Ensemble Methods

In contrast to the ordinary learning methods which try to train a single base learner, Ensemble learning methods train a combination of base learners to strengthen the ability of prediction. Base learner in ensemble methods can be any machine learning algorithm such as decision trees. Ensemble learning includes two general methods which are Averaging methods and Boosting methods. In averaging methods, several base learners are trained independently and the average of their predicted values is the prediction of the averaging method. In this method the prediction error can be reduced by reducing variance. Random Forest and Bagging method are the famous examples of this method. The other ensemble method is Boosting which starts with a weak base learner and add a new one iteratively to produce a stronger learner. Prediction error of boosting methods can be reduced by reducing bias. AdaBoost and Gradient Boosting are the subsets of boosting methods. In this project, regression tree is selected as the base learner also, Random forest and gradient boosting have been used as the ensemble methods on the dataset.

## 2.3. Random Forest

Random Decision Forest is an ensemble learning method which can be used for classification and regression tasks [3]. This method has been developed to improve prediction accuracy of decision trees, overcome overfitting issue with decision trees and be more robust to noise. To achieve this purpose, random decision forest, as an ensemble method, tries to decorrelate the collection of decision trees by applying them on random samples of training data and also using randomly selected features for training trees [4].

In this method, first, bootstrap sampling operates on training examples with replacement for k times, therefore we would end up with k samples with the same size which they are drawn independently from the same probability distribution, in other words they are IID. On each sample, a decision tree is trained based on random selection of features for splitting each node of tree. Each tree outputs a prediction, the majority vote of these predictions (for classification) and the average of the predictions (for regression) would be the predicted value of random forest. In other words, random forest aggregates the predictions of all trees (Figure 2). In this project, random forest regression based on regression trees, has been developed on the dataset. Hence, $K$ different regression trees can be trained on $K$ IID samples drawn from training set with replacement and the result of random forest would be,

$$f(\mathbf{x}) = \sum_{i=1}^{K} \frac{1}{K} f_i(\mathbf{x}) \quad (9)$$

where, $f_i$ is the i$^{th}$ tree.

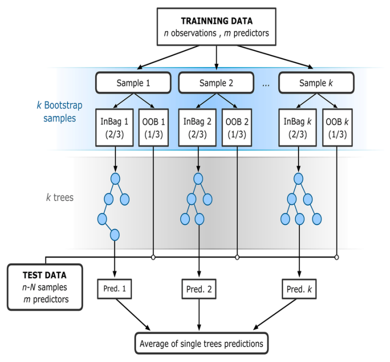

Figure 2. Random Forest Flowchart for regression [5]

## 2.4. Gradient Boosting

Gradient boosting falls under boosting methods of the ensemble learning family. The first boosting algorithm was AdaBoost (Freund and Schapire, 1997), which was originally created for binary classification. Then Leo Breiman proposed that boosting can be looked at as an optimization problem, which was followed Jerome H. Friedmans gradient boosting algorithms for regression problems.

The motivation behind boosting was to leverage weak learners to produce a strong predictor. These weak learners are defined to have a performance only slightly better than random guessing. The goal of boosting is to sequentially add weighted predictions of the weak learners together so that more accurate learners have more influence on the final prediction.

Boosting methods require a weak learner, an additive model and a loss function. Our interest lies in Gradient Boosting, which applies the concept of gradient descent to boosting. It uses a forward stage-wise additive model to minimize a loss function chosen based on the type of problem at hand. The loss function could be least squares for regression problems and log loss for classification problems but in any case it should be differentiable. The weak learners in Gradient Boosting are small regression trees.

There are several regularization strategies to prevent overfitting with Gradient Boosting. Generally, more trees along with small trees or stumps have been proven to slow overfitting and have better results. Additionally, introducing shrinkage can improve the model. This strategy involves scaling each update by a learning rate. Decreasing learning rate (Shrinkage) slows down learning the data because more trees need to be added to the model. [6] demonstrates that common values of learning rate include values less than $0.1$ and values between $0.1$ to $0.5$. Another regularization strategy is called stochastic gradient boosting, in which we randomly subsample without replacement from the training set to fit a regression tree at each iteration. By fitting a learner to a smaller dataset, the algorithm learns faster. Subsampling is beneficial because it introduces randomization into learning. However, it limits the amount of data available for training.

[6] develops the theory behind gradient boosting and several different algorithms for specifying a loss function for the general algorithm. For example, LSBoost assigns least squares to the loss function. It discusses regularization strategies like shrinkage along with its tradeoff with number of trees. [7] demonstrates a minor modification to the gradient boosting algorithm by introducing randomization.

Gradient Boosting follows an additive model by fitting a weak learner at each iteration that minimizes a loss function, *i.e.*,

$$F(\mathbf{x}; \{\beta_m, \mathbf{a}_m\}_1^M) = \sum_{m=1}^{M} \beta_m h(\mathbf{x}; \mathbf{a}_m), \tag{10}$$

where $\beta_m$ are the expansion coefficients, $\mathbf{a}_m$ are the learner parameters and $h(\mathbf{x}; \mathbf{a}_m)$ are the weak learners. To minimize the total loss function, ”greedy-stagewise” approach is used, *i.e.*,

$$(\beta_m, \mathbf{a}_m) = \arg\min_{\beta, \mathbf{a}} \sum_{i=1}^{n} L(y_i, F_{m-1}(\mathbf{x}_i + \beta h(\mathbf{x}_i; \mathbf{a})), \tag{11}$$

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \beta_m h(\mathbf{x}; \mathbf{a}_m). \tag{12}$$

To minimize each stage's loss function, first we calculate the negative gradient which defines the steepest descent direction,

$$-g_m(\mathbf{x}_i) = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}. \tag{13}$$

Then we fit a learner that best approximates that value,

$$\beta_m, \mathbf{a}_m = \arg\min_{\beta, \mathbf{a}} \sum_{i=1}^{n} [-g_m(\mathbf{x}_i) - \beta h(\mathbf{x}_i; \mathbf{a})]^2, \tag{14}$$

where the step size $\beta_m$ is found through steepest descent. Finally, we use the step size and the best learner to update our previous estimator.

As mentioned above, overfitting can be controlled by introducing shrinkage. This adjusts the equation at the update step, by scaling the step size $\beta_m$ by a factor of $0 \leq v \leq 1$, *i.e.*,

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + v\beta_m h(\mathbf{x}; \mathbf{a}_m). \tag{15}$$

## 3. Implementation

In this section we describe the dataset, its preprocessing and our implementation.

## 3.1. Dataset and Preprocessing

In this section we describe the main dataset that we used to test our methods. We'll start by describing the raw dataset and then we explain the preporcessing of the data.

This dataset is named *House Sales in King County, USA* was chosen from *kaggle.com* [1] and includes 21613 datapoints with 21 features. Some of these features don't have a linear relationship with the house price such as 'date', 'long' and 'lat' representing the date the house was sold, the longitude and the latitude of the house, respectively. These features should either be removed or modified. Inspired by Kernels [8] [9] found on kaggle website, the following preprocessing steps were applied on the data:

First, using 'date' (date the house was sold) and 'yr_built'(the year the house was built), we calculate the age of the building. Using the feature 'yr_renovated' (year the house was renovated) we create a new binary feature to represent whether the house was renovated at all. This was done since for the houses without renovation, 'yr_renovated' is zero.

Although zipcode doesn't have a linear relation with the price, it could have useful information about the house price. Hence it is treated as a categorical feature and is replaced by a vector of all zeros except one 1 at the corresponding zipcode.

Next, the features 'id', 'date', 'yr_built', 'lat', 'long', 'date_yr' and 'yr_renovated' are removed.

We take log of the price and features with skewness of more than $0.75$. This results in the features having a normal distribution (Figure 3).

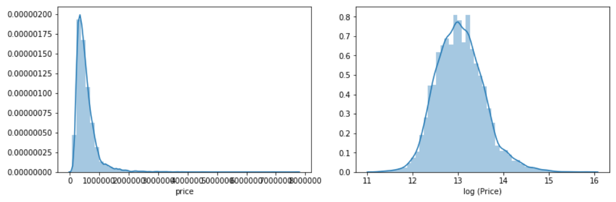

Figure 3. Histogram of Price and log(Price)

Then, the data was normalized so that each feature has values between 0 and 1. Finally, the data was split into training and test data. As a result, the preprocessed data has 85 features, 14480 training points and 7133 test points.

## 3.2. Linear Regression

Linear Regression was implemented in Python, using *sklearn.linear_model* package. For Ridge regression, LASSO, and Huber robust regression, 4-fold cross validation was used to determine the best value of $\alpha$ for regularization strength. Then the 20 most important features of each method, *i.e*. the features with the largest absolute value of coefficients, were plotted. Finally, performance metrics Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) and R-2 score of each method were calculated and compared. Moreover, training run-time for each algorithm was measured.

## 3.3. Random Forrest

For implementing random forest regressor, *SkLearn* package in python have been used. This package consists a number of ensemble modules and we will use a class named *RandomForestRegressor*.

In the ensemble, each sample is drawn with replacement according to bootstrap sampling method from training set and each tree is built on that sample. In this class, splitting a node is not based on the best split among all features. Instead, selected split is the best split among a random subset of the features. After training, each tree is applied to a test point. In other words, test point falls in the one of partitioned region from training stage and the predicted value for test point is the average of all training values in that region. Predicted value for test point is average of predicted value of all trees. As a result, because of averaging, variance of predicted value of random forest decreases. This model is specified by some parameters that the most importance ones are as follow.

*n_estimators* which represents number of trees. *Criterion* represents measure of quality of a split. Mean square error is considered as the criterion. *max_features* is the number of features that should be considered when looking for the best split. *Max_depth* represents the maximum depth of the tree. random state is the seed that is used by random number generator. By running this class we can design a random forest regressor. Next step is training random forest with training data and then apply it over test set and predict the value of test set.

To explore the impact of these parameters, random forest regressor has been trained on training data by varying each parameters and then it has been applied on test data.

To obtain the optimum value for tuning parameters, 5-folds cross validation has been used on the training data. 700 for number of trees, 68 for *max_features* and 40 for *Max_depth* were found to be optimum. After obtaining tuning parameters, random forest regressor can be constructed based on these parameters and would be trained on training data. At the end, random forest has been applied on the test set and multiple performance metrics have been used to evaluate the performance of the model.

## 3.4. Gradient Boosting

We developed the model in MATLAB using the *fitensemble* function and specifying LSBoost as the method and equivalently the *fitrensemble* function to create a model of

[1] www.kaggle.com/harlfoxem/housesalesprediction/data

learners for regression. The model allows you to specify number of boosting cycles (or number of estimators), tree parameters, learn rate, resample fraction, cross-validation options, etc. Tree parameters include maximum number of splits (*MaxNumSplits*) and minimum leaf size (*MinLeafSize*). When including resample fraction to test model with subsampling for stochastic gradient boosting, the *Replace* parameter must also be set to *off*; because we are interested in subsampling, which means resampling without replacement. When conducting cross validation, cross-validation parameters are set and the *kfoldLoss* function is used to find the corresponding k-fold MSE loss. To predict the response of the LSBoost model, the predict function is used.

## 4. Results

### 4.1. Linear Regression

By observing Figure 4, we can see that the effect of cross-validation on the training RMSE, for Huber is very small relative to Ridge and LASSO.

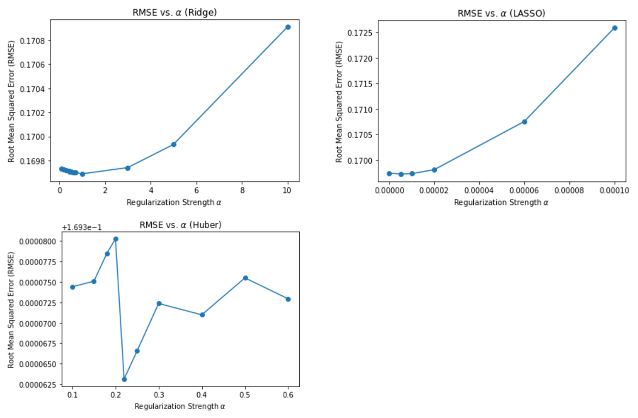

Figure 4. RMSE versus Regularization Strength $\alpha$ in Cross Validation for Ridge, Lasso and Huber, respectively.

Looking at Figure 5, we observe that in OLS, there are a few features with very large coefficients. However, in other regularized methods, the coefficients are distributed among more features. We also note that zip-codes are among the important features to predict a house price.

Comparing the performance metrics of each method (Table 2), we notice that OLS, Ridge and LASSO have performed very similarly. OLS has performed the best in RMSE and R-2 score, and the worst in MAE, where Huber has performed the best.

### 4.2. Random Forest

As it has been mentioned in implementation, effect of each tuning parameter on the random forest model need to be explored. Figure 6 Top Left, shows variation of RMSE and R-2 score for both training and test data by increasing number of trees. We can see that it when number of

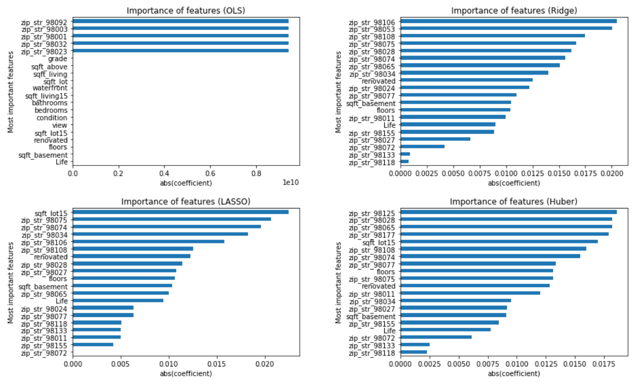

Figure 5. 20 most important features and their values for each method.

| | MAE | RMSE | R-2 Score |
|---|---|---|---|
| OLS | 0.0288390 | 0.0394235 | 0.867667 |
| Ridge | 0.0288344 | 0.0394237 | 0.866697 |
| LASSO | 0.0288338 | 0.0394262 | 0.866982 |
| Huber | 0.0287275 | 0.0394678 | 0.864843 |

Table 1. Performance metrics of different methods

trees are increasing RMSE for training and test data reduces and for R-2 score case, it increases by increasing number of trees. Figure 6 Top Right and Bottom, show the variation of RMSE and R-2 score by varying tuning parameters, Max Features and Max Depth, Respectively. From these three plots we can conclude our model does not overfit.

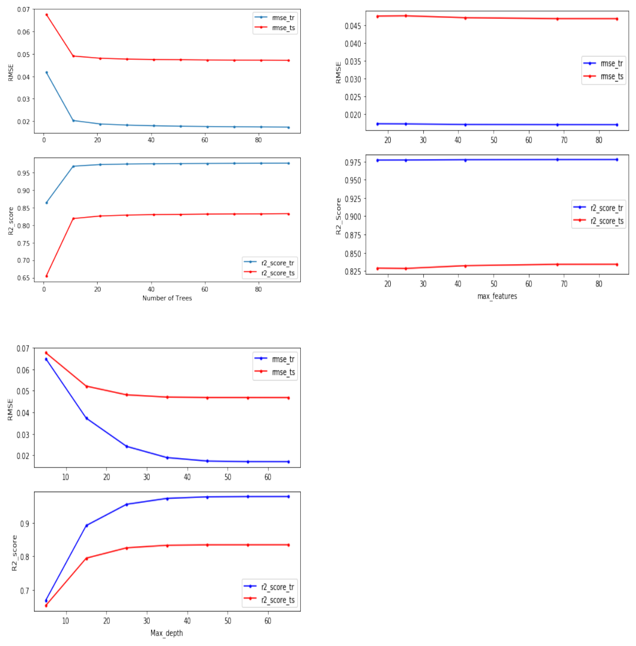

Figure 6. Variation of training and test score by varying number of trees (Top Left), Max Features (Top Right) and Max Depth (Bottom)

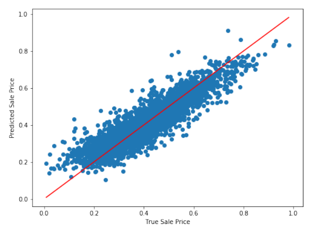

Figure 7. Predicted Price versus True Price

Predicted sale price versus true sale price has been plotted in Figure 7. The red line is the ideal case in which the predicted values are equal to the true values and we see that scatter plot is very close to it.

### 4.3. Gradient Boosting

First, we calculated the 10-fold cross validation generalization error which estimates the performance of the model on unseen data. It demonstrated that the MSE error levels out at less than $0.04$ after $200$ boosting cycles. That measure also tends to underestimate the actual test error. We then plotted the MAE, MSE, RMSE and R-2 metrics of the training data and test data for fewer $150$ boosting cycles. The model still performs well on the training data, but hints at over-fitting for test data. So we performed cross-validation over the number of trees and shrinkage parameters and found that the error decreases until $0.1$ and then increases again after $0.3$. Therefore, optimal values of learning rate are between $0.1$ and $0.3$. Using value a learning rate of $0.1$, we again computed the metrics, for the training and and test set and found that the training error and test error become the same for $150$ boosting cycles and no significant overfitting appears to occur (Figure 8).

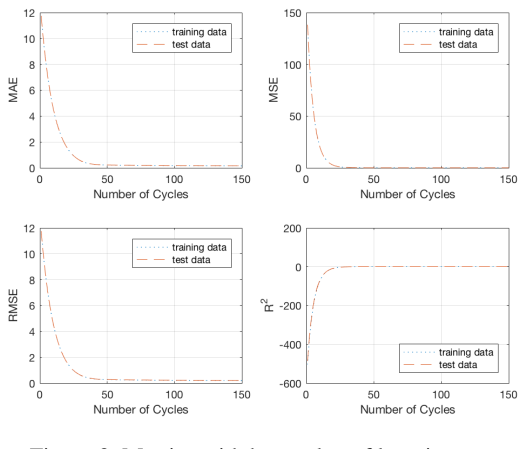

Figure 8. Metrics with best value of learning rate

We also wanted to test how the tree parameters affected the model. So we performed cross-validation over *MaxNumSplits* and *MinLeafSize*. We found that the cross-validation error increases with increasing *MaxNumSplits*, but different values of *MinLeafSize* did not have significant influence on the model. This is what we expect because gradient boosting adds over weak learners so a smaller number of splits does not allow the regression tree to grow long branches and thus, keeps the weak learners weak (Figure 9).

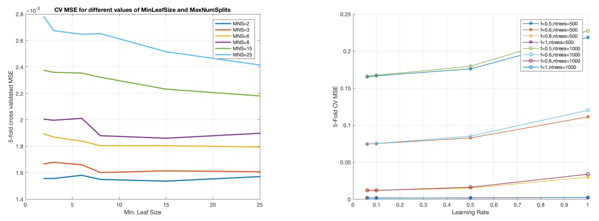

Figure 9. Cross-Validation MSE Results versus MinLeafSize for different MaxNumSplits (left) and LearningRate for different Subsampeling Fraction and Number of Trees (Right)

To determine the effect of subsampling, we again performed cross-validation on the number of trees, learning rate and resample fraction. Figure 9 demonstrates that no resampling (fraction=1) had the best results and that neither number of trees equal to $500$ nor $1000$ showed a significant difference in performance. The error was at its minimum with learning rate equal to $0.1$. These results were not what we expected because subsampling reduces the correlation between trees, thus reducing the variance. These results also didnt change when we normalized the dataset.

## 5. Conclusion

In this project, we implemented three methods to solve a regression problem. Additionally, we learned how to understand our data by performing strategies for preprocessing. For example, if features dont have a linear relationship with the target variable, then it is possible to remove that feature from the data. For linear regression, normalizing the data improved results, while for the other methods, the values of the results were overall lower, but the trend of the results were unchanged. Ultimately, Gradient Boosting produced the lowest errors. Even though Linear Regression seems very simple, it can achieve good results and computes in a short amount of time. On the other hand ensemble methods take a lot longer to compute, because Random Forests grows trees with long branches and Gradient Boosting boosts over a lot of trees, especially with shrinkage. The time complexity of learning each tree in the ensemble methods is $O(n\log(n))$ [10].

## 6. Description of Individual Effort

Each team member individually conducted his or her owns methods of preprocessing the larger data set. After we did this exercise, we combined all of each others strategies together to produce the final preprocessed data. Artin focused on the methods and implementations of linear regression. Mehrnoosh focused on Random Forests and its implementation. Gabrielle focused on Gradient Boosting and its implementation. Each team member wrote his or her part of each section of the report that corresponded to his or her focus. Additionally Artin wrote section with preprocessing.

## References

[1] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[2] Wikipedia. Linear regression. [Online; accessed 1-Oct-2017].

[3] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[4] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[5] Victor F Rodriguez-Galiano and Peter M Atkinson. Modelling interannual variation in the spring and autumn land surface phenology of the european forest. *Biogeosciences*, 13(11):3305, 2016.

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

[7] Jerome Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[8] Pedro Marcelino. Comprehensive data exploration with python. [Online; accessed 1-Nov-2017].

[9] Alexandru Papiu. Regularized linear models. [Online; accessed 1-Nov-2017].

[10] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

## 7. Appendix

Here we have included the additional figures and information.

The codes are uploaded on the *blackboard*. The data preprocessing, linear regression and random forest are implemented on Python, separately. Gradient Boosting is implemented using MATLAB. The python codes include comments in the code itself. There will also be a README file provided to describe the MATLAB Codes.

In the following table, the performance metrics of the 3 main algorithms are compared.

| | MAE | RMSE | R-2 Score |
|---|---|---|---|
| OLS | 0.0288 | 0.0394 | 0.867 |
| Random Forrest | 0.0336 | 0.0468 | 0.834 |
| Gradient Boosting | 0.0275 | 0.0383 | 0.889 |

Table 2. Performance of different Algorithms

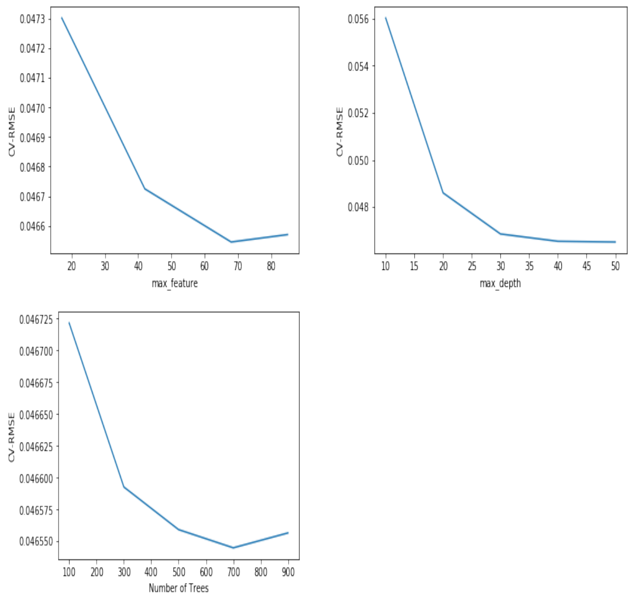

Figure 10. Cross validation for Tuning Parameters (Random Forest)

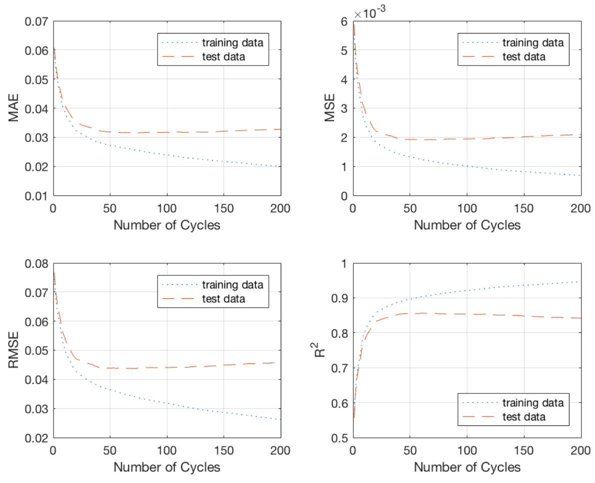

Figure 11. Metrics of training and test data without shrinkage (Gradient Boosting)