Beating the Bubble: Predicting House Prices in Ames, Iowa

Jesus Cervantes, Alex Kim, and Alexandre Bucquet

June 12, 2018

1 Motivation

In this project, we aim to maximize the effectiveness of machine learning in predicting housing prices. Because the house prices depend on a multitude of interrelated features whose true importance is difficult to gauge, we feel that this project provides a good opportunity to explore machine learning beyond the standard techniques. Furthermore, we hope to improve upon past work by considering groupings of similar houses, both geographically and based on other features.

2 Task Definition

More specifically, we will design a model whose input is a set of features concerning a house and its sale, and whose output is the house's predicted sale price. At the end of the project, we expect to be able to closely approximate the sale price of every house in Ames, Iowa; we also hope that the techniques used here can be easily transferable to other cities in the United States or even the world in general.

Our base dataset comes from Kaggle[1], and was posted as part of a competition that challenged users to creatively approach prediction of house prices. The data includes a list of 1,460 house sales that took place in Ames, Iowa; each entry contains 79 house and sale features as well as the final sale price for the house (see Table 1). To improve our prediction, we will also collect spatial data on Ames, IA. We intend to find welfare and demographic data on City Data[2], and will also include information about school proximity and bus lines access for every house in our dataset.

Land Slope	**Utilities**	**Sale Month**	**Sale Year**	**Sale Type**	**Sale Condition**	**SalePrice**
Gtl	AllPub	2	2008	WD	Normal	208500
Gtl	AllPub	5	2007	WD	Normal	181500

Table 1: Sample Input-Output Pairs.

Note: for convenience, we are illustrating only a small portion of the 79 features our dataset has.

3 Survey of Related Work

Little academic literature has been published specifically regarding machine learning applications on house prices; however, we managed to find an interdisciplinary paper[3] by professors of the Department of Economics and the Department of Computer Science at New York University. The paper, "Machine Learning and the Spatial Structure of House Prices and Housing Returns" by Caplin et al., specifically focuses on housing data from Los Angeles, which includes both exact locations and sale histories of each house. Building

[1] `https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data`
[2] `http://www.city-data.com/races/races-Ames-Iowa.html`
[3] `https://libguides.uwf.edu/ld.php?content_id=42292186`

upon existing literature, Caplin et al. apply standard regression techniques; however, to expand, they also pay closer attention to neighborhoods of houses. More specifically, they divide the houses into neighborhoods based on both municipalities and census blocks, and then run regressions on these local neighborhoods. As a result, the predictions are more granular and benefit from increased accuracy.

We were unable to find additional literature pertaining specifically to machine learning on housing markets; however, we found special interest in Caplin et al.'s use of neighborhoods.

To gain a deeper look into machine learning, we also approached James et al.'s textbook *Introduction to Statistical Learning* (*ISLR*). While introductory, the textbook provides a broad survey of machine learning techniques, allowing us to explore beyond the topics covered in class. We took special interest in weight regularization, gradient boosting, and best subset selection, as each seemed to provide a new, feasible technique we could develop iteratively. *ISLR* also takes a more detailed look at cross-validation, which allows for more robust evaluation of models.

To learn more about boosted gradients, we looked into the original paper by Stanford professor Jerome H. Friedman, "Stochastic Gradient Boosting."[4] In this article, Friedman provides an overview of gradient boosting techniques for regression, and also explores how adding randomization to the procedure can improve the model's performance on test data. Rather than updating the model using the full training data, using a randomly selected subsample of the training data can be used instead to prevent overfitting.

Furthermore, to gain a deeper understanding of the lasso, we explored Stanford professor Robert Tibshirani's paper "Regression Shrinkage and Selection via the Lasso."[5] This article provides a more rigorous mathematical background of the technique, as well as a glimpse into other uses outside of regression. We paid special attention to the exampe of lasso regression on prostate cancer data. Since the paper contains a visualization of the coefficients as well as numerical values, we were able to gain a stronger intuition of how the technique works.

4 Infrastructure

To experiment with each of our machine learning techniques, we developed a standardized pipeline to feed data to each method and test its performance, as well as save its output.

To prepare our data for testing and evaluation, we cleaned up certain features in our original .csv file from Kaggle using a R script (see feature extraction in section 5.2 for more information). During this initial step, we also dividided the data into our training and test partitions, each its own .csv file. From there, we ran the training .csv through the Python library *Pandas* to extract example tuples for training our models. Each of these tuples is of length 2, where the first value is the feature vector and the second is the true sale price.

To facilitate cross-validation, we then split the training data into randomized training and validation subsets, with roughly a 9:1 ratio. For each partitioning of the training data (10 total), we trained our specific method for prediction (k-means, linear regression, etc.) and stored the results in a *.p* file using the *Pickle* package, as to save us from running the same computation multiple times (i.e clustering is redundant when we want to adjust how we run linear regression on each cluster since the clusters stay the same).

Additionally, we fed the training data and each model and into an evaluator function, which returns the mean squared loss of the predictor on the given dataset. This evaluator function allowed us to assess the training and validation error of our models for each iteration of cross-validation. These error rates allowed us to tune our hyperparameters to maximize generalizability of our models.

[4] `http://statweb.stanford.edu/~jhf/ftp/stobst.pdf`
[5] `http://statweb.stanford.edu/~tibs/lasso/lasso.pdf`

5 Approach

5.1 Proposed Methods

In attempting to solve this problem, we first set out to implement a linear regression model trained on our features for each house. As covered in class, we optimized our weights using stochastic gradient descent. We tested the effectiveness of this model using our cross-validation function before we began work on more complex models to improve upon the original.

5.2 Feature Extraction

Dealing with such a large number of different kinds of features, it was impossible to simply upload the .csv file into data structures and run logistic regression immediately. There were several cases that needed to be handled individually in order to make the data usable. We used R to address some of the issues in the data.

First, not all data points had data available for every feature. In the case of numerical entries (i.e. garage age), we were simply able to replace N/A with 0. For some features, where we felt this might tamper with the regression, so we manually modified the feature values. This concerned primarily features such as “Year Garage Built,” where replacing N/A’s with 0 would give us many data points with values near 2000 and a nontrivial subset with values of 0. To address this, we modified the data template to instead be of the form “Garage Age,” replacing the replacing “Year Sold” and “Year Garage Built” with a singular parameter whose value is “Year Sold” - “Year Garage Built.”

For enumerated string-valued features, such as “Lot Shape,” we had initially thought to implement a *word2vec* protocol. However, we instead decided to replace each of these features with a series of indicator features, one for each enumerated choice. This enabled us to isolate the presence of each lot type and its impact on the price of our house. In order to prevent our data from getting too sparse as a result, we implemented our algorithms using dictionaries rather than lists. After implementing the previous operations as well as getting rid of pointless features (i.e “Street” which had a value of “Pave” for all data points in our dataset), our basic dataset from Kaggle was ready to train models.

After running several algorithms, we decided to find more data to aid with a neighborhood-based approach. In our original dataset, each house has a “neighborhood” feature[6]. We augmented our dataset by adding nine new neighborhood-specific fields (see Table 2): Median House Income, Median Age, Unemployment Rate, Percentage of Whites, School Access, Bus Access, Mean House Price, Median House Price, and Standard Deviation of House Price. The first three data pieces were extracted from City Data[7]. School Access and Bus Line Proximity are measured by number of schools/bus lines in the neighborhood (from Google Maps). Finally, data on house prices was compiled in R according to our training set.

Neighborhood	Median Income	White %	Schools	Median Age	Mean House Price
Blmngtn	89.286	96.3	3	37.5	194.8708824
Blueste	55.2	72.6	1	30.2	137.5

Table 2: Sample Neighborhood Data

Note: for convenience, we are including only a 5 of the 9 augmented features our dataset contains.

[6] There are 25 neighborhoods in the data that can be visualized here: `https://drive.google.com/a/bucquet.com/open?id=1GV1ZA27fJbYTuxvOOqmseTetXEZ4fyHt&usp=sharing`

[7] `http://www.city-data.com/races/races-Ames-Iowa.html`

5.3 Preliminary Methods: Baseline and Linear Regression

To provide a baseline comparison for our gradient boosting method, we computed a simple baseline predictor based on the median sale price. We also implemented a standard linear regression to provide context for the effectiveness of linear regression with boosted trees. Detailed information on these methods and their results can be found in Sections 6.2 and 6.3, respectively.

5.4 Gradient Boosting

The primary goal for this model is to provide predictions that are more accurate than simple linear regression. To understand our gradient boosted approach, it is easiest to first consider the standard linear regression, in which the objective function we attempt to minimize is:

$$\sum_{i=1}^{n}(\phi(x_i)w - y_i)^2$$

In our linear regression, we use stochastic gradient descent to find the weight vector w that minimizes the above formula in which x_i is one input, $\phi(x_i)$ is the processed input, and y is the correct output.

On the other hand, in gradient boosting we first train a linear regression model, then train a second model on the error of the first. Let's define the following:

$$E_i = (\phi(x_i)w - y_i)^2 \quad i \in \mathbb{N}$$

Where we set w to be the weights obtained after the linear regression fitting. Then we can train a second linear regression model with weights w' using the updated objective:

$$\min_{w'} \sum_{i=1}^{n}(\phi(x_i)w' - E_i)^2$$

This is effectively training a model to predict the error of our original model. We nested this process into a for loop enabling us to control the number of boostings (models) we use in our predictor. For N iterations, our prediction with gradient boosting is:

$$P(x_i) = \sum_{j=1}^{N} \phi(x_i)w_j, \quad i \in \mathbb{N}$$

Where w_j is the weight vector trained be the j-th iteration of the linear regression.

5.5 Clustering

To further improve our predictions, we attempted to cluster the data in order to get more focused and precise models. Indeed, the sale prices in our dataset span a wide range, and splitting the data into groups could help us alleviate this issue. As reference, the minimum sale price in our dataset was \$34,900, while the most expensive house was sold for \$750,000.

In all of the following models, we used the k-means algorithm that assigns data points to k clusters in order to minimize the distance of every point to its assigned centroid. More formally, if we have n observations $x_1, ..., x_n$, we want to find $\mu_1, ...\mu_k$ to minimize:

$$\sum_{i=1}^{n}(x_i - \mu_j)^2$$

where μ_j is such that $(x_i - \mu_j)^2 \leq (x_i - \mu_c)^2$ for all $c \in \{1, ..., k\}$.

After splitting the data in k groups, we trained a boosted regression on each individual cluster. Then, our final predictor first identifies which cluster the input belongs to, and then applies the corresponding learned boosted predictor for that group.

More formally, our predictor is:

$$P(x) = \Phi(x)w_j$$

where w_j is the weight vector for cluster j's predictor, and j is such that $(x - \mu_j)^2 \leq (x - \mu_c)^2$ for all $c \in \{1, ..., k\}$.

House Clustering

First, we wanted to split the houses into cluster based on their characteristics, to see more specific predictors based on house features would improve our predictions.

We also tried replacing the first linear regression in our boosting by the sale price of the cluster the house was assigned to. We will refer to this second attempt as boosted k-means in the rest of this paper.

Neighborhood Clustering

Finally, we attempted to group the houses by neighborhood. We ran the k-means algorithm on the neighborhood data we collected, and then split the houses up based on neighborhood cluster assignments.

5.6 Ridge Regression & Lasso

Our final model attempted to reduce overfitting. To do so, we can add a penalty depending only on w to the loss function we are optimizing. Namely, our linear regression now minimizes

$$\sum_{i=1}^{n}(\phi(x_i)w - y_i)^2 + \lambda f(w)$$

where $f(w)$ is the penalty, and λ is an independently specified "tuning coefficient."

For a ridge regression, we define $f(w) = ||w||_2^2$ (the squared L_2 norm of w), thus the objective function is:

$$\min_w(\sum_{i=1}^{n}(\phi(x_i)w - y_i)^2 + \lambda||w||_2^2)$$

By including the L_2 norm in the objective function, optimization of the weight vector results in relatively small weights. By keeping the weights small, we aimed to reduce any extremities in the model, allowing it to generalize better.

For lasso regression, we define $f(w) = ||w||_1$ (the L_1 norm of w), thus the objective function is:

$$\min_w(\sum_{i=1}^{n}(\phi(x_i)w - y_i)^2 + \lambda\sum_{i=1}^{m}|w_i|)$$

In contrast to ridge regression, lasso regression completely eliminates certain weights by nature of optimizing the L_1 norm. We chose to experiment with lasso to examine the possibility that certain features might not affect house prices at all, and instead only introduce noise and overfitting.

6 Results

6.1 Metric

To assert the quality of our individual prediction, we use the standard squared deviation:

$$\text{Loss}_{\text{squared}}(w, x, y) = (w\Phi(x) - y)^2$$

We use the mean squared loss on all data points to evaluate each predictor.

Another way to assert the soundness of our model is to use the coefficient of determination, or R^2. This coefficient encapsulates how much of the variability our prediction is explained by the variability with the data:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(P(x_i) - S_i)^2}{\sum_{i=1}^{n}(\mu_S - S_i)^2}$$

where n is the number of data points, μ_S is the mean sale price of the dataset, S_i is the price of house i, and $P(x_i)$ is our prediction for the price of house i.

6.2 Baseline

As a naive baseline predictor, we predicted each house's sale price to be the median of all sale prices in our dataset. To assess the accuracy of this baseline estimator, we computed its mean squared error over the all true sale prices, with a mean squared error of $12,378.501$ (thousands of dollars squared). This corresponds to an average deviation of about \$110,000 on each house.

6.3 Linear Regression

As a simple model, we performed linear regression on the entire dataset. As expected, this yielded significantly better results than our naive baseline predictor, with a mean squared error of $2,286.54$ (thousands of dollars squared). This corresponds to an average deviation of about \$48,000 on each house.

6.4 Linear Regression with Gradient Boosting

For both the linear regression and the gradient boosting, we separated our dataset into two parts. We allocated around nine tenths of our dataset for training and validating the models, and the other tenth to perform our final model evaluation. To better assert the quality of our models, we performed cross-validation by repeatedly leaving out a different tenth of the training data and using it for validation (see Algorithm 3 in Appendix).

We ran our predictor on 1 to 5 trees, using 500 iterations for stochastic gradient descent, and 10 data splits. The results of this experiments are shown.

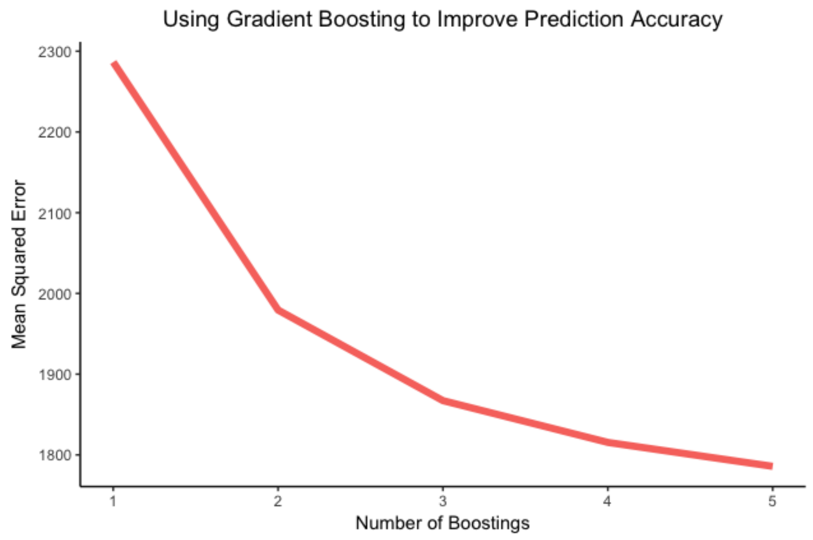

Figure 1: Using Gradient Boosting to Improve Prediction Accuracy

The above graph clearly shows the advantage of using gradient boosting over a plain linear regression: using two trees already improves the average squared error by more than 500 thousand dollars squared (from 2,286.54 to 1785.816), which represents an improvement of around 22%. In terms of accuracy of prediction, our mean error drops from \$48,000 with normal linear regression to \$42,250 with 5 trees. It is interesting to note that adding more trees doesn't improve our performance linearly, which seems reasonable since the closer our predictor is to the actual answer, the harder it is to improve.

In the future, we could attempt to run more extensive tests to compare the hyper-parameters (number of boostings, step size, number of iterations for gradient descent, etc.) to understand the performance of gradient boosting more thoroughly. We expect gradient boosting to have a fast rate of convergence to a precise predictor which means that less iterations of stochastic gradient descent might suffice to obtain a good predictor.

6.5 Clustering

The effects of clustering on the accuracy of our predictions were very mitigated. While some pairings of number of clusters and type of grouping performed similarly to the boosted regression, other choices undermined the accuracy of our predictor. More specifically, too many clusters produced overfitting on our dataset, and thus significantly increased the test error. Detailed results can be found in the table below:

House Clustering (4)	Boosted k-means (15)	Neighborhood Clustering (2)	NC (6)
6758.89	6009.944	2147.58	3661.836

Table 3: Test Error for Clustering Methods

Note: All models are described using the following format: *Model name (number of clusters)*

Although these results seem to show that clustering is not an adequate choice, it is important to note that the results for neighborhood clustering (NC) were obtained with only 50 iterations of gradient descent and 3 gradient boostings. This means that when the number of groups is moderate, using a clustering method produces results that are more accurate than the normal gradient boosting approach.

6.6 Ridge & Lasso Regularization

As with our other linear regression models, we implemented cross-validation in our ridge and lasso regression. Our results from ridge regression were fairly straightforward; the reduced weights did indeed allow the model to generalize better, as it reduced overfitting by reducing weights and/or removing some features entirely.

In the end, our model using the ridge regularization obtained a mean squared error of 1690.66, which corresponds to an average deviation of \$41,100 on house price prediction. Our lasso regularization had a mean squared error of 1498.02, which translates to an average error of around \$38,700 on price prediction.

7 Discussion

First, we observe that gradient boosting provides a sharp increase in accuracy compared to the linear regression. More specifically, it improves the R^2 metric from 0.638 for a plain linear regression to 0.717 for five iterations of gradient boosting.

Clustering, although the fastest converging method, is more prone to overfitting, hence one needs to be careful in choosing the number of clusters to use. More specifically, the clustering methods – house clustering, boosted k-means and neighborhood clustering – obtain an R^2 metric of around 0.5. However, the best model (neighborhood clustering with two clusters) had an R^2 of 0.660; this is the best computing speed ti accuracy ratio of all of our models.

Finally, feature selection with either lasso or ridge regularization can further improve predictions by up to 20%. These two approaches have an R^2 of 0.763 and 0.732, respectively.

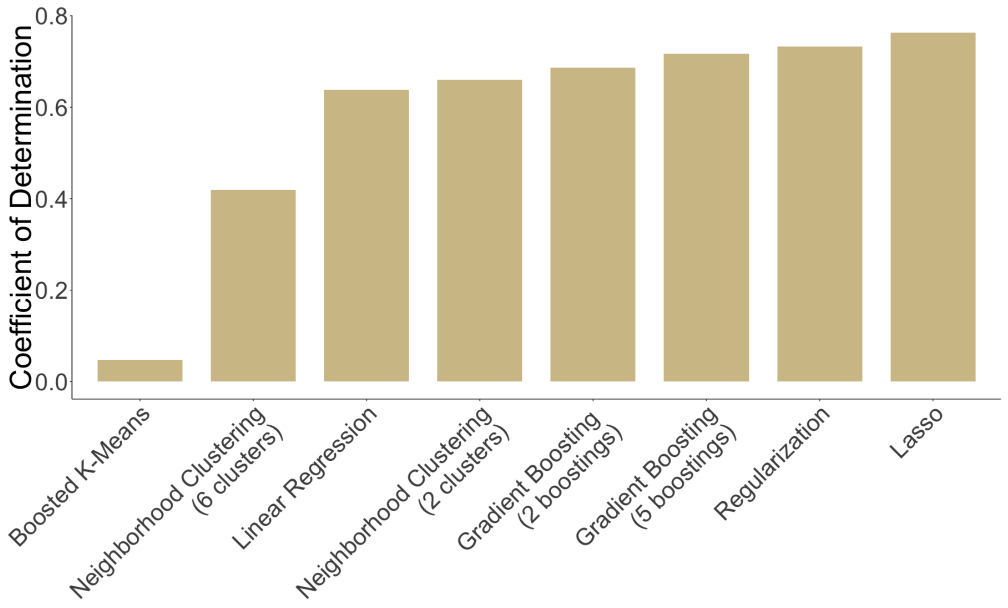

Figure 2: Comparison of R^2 values

8 Future Research

8.1 Further Experiments

In the future, it might be worthwhile to quantify the impact a variable has on the house price by looking at the correlation coefficients between the house price and the particular feature. Formally, if X is the feature and S is the sale price, we define the correlation coefficient $p_{X,S}$ to be:

$$p_{X,S} = \frac{E[XS] - E[X]E[S]}{\sigma_X \sigma_S}$$

Where σ_X and σ_S are the respective standard deviations of X and S and $E[X]$ and $E[S]$ are the respective means of X and S.

To evaluate the value of each feature, we propose to use the adjusted coefficient of determination for models that have and don't have the feature we would like to evaluate. Using the adjusted coefficient of determination also accounts for k, the number of features we use, thus provides a fairer measurement to evaluate the importance of a feature if we remove it.

$$R^2_{\text{adj}} = 1 - (1 - R^2)\frac{n-1}{n-k-1}$$

This would allow us to perform a best subset selection, which can complement the feature selection performed by the ridge regression and the lasso.

We would also like to further investigate the benefits of adding more than five trees; to quantify the tradeoffs, we could analyze runtime, prediction precision and degree of overfitting. This is merely a matter of acquiring more computing power.

8.2 New Models

To further improve our predictions, we can consider to account for the relationship between variables by adding non-linear features to our model. Since we already have 79 features per data points, we would be sure to use some regularization method to minimize the overfitting of our model. In a similar vein, another possible model would be a simple neural network that generates additional features, namely the squared values and cross-terms of the numerical features.

To speed up the computation time of our current algorithms, we can also batch some of the gradient descent iterations, which would be particularly useful if we were to post our predictors online.

9 Conclusion

Having experimented with a variety of machine learning technqiues, we were able to attain maximum testing accuracy using lasso regularization. While this makes sense, given that a model with less features is more generalizable, we remain curious to see how our methods would perform given a wider range of hyperparameters and more computing time.

Because our dataset did not include the specific address of each house, we were unable to track with complete certainty which sales pertained to the same house. However, given the feature history and exact geographic location of each house, we remain curious to see what additional advances can be made with machine learning on housing markets.

References

[1] Trevor Hastie Gareth James, Daniela Witten and Robert Tibshirani. *An Introduction to Statistical Learning*. Springer, 2013.

[2] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society*, 58(1):267–288, 1996.

[3] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 02 2002.

Appendix – Algorithms

This section contains pseudocode for linear regression using stochastic gradient descent (Algorithm 1), gradient boosting (Algorithm 2), and cross-validation (Algorithm 3). Our full code can be found on GitHub[8].

Algorithm 1 Stochastic Gradient Descent

```
1: procedure LEARNPREDICTOR(EXAMPLES, NUMITERS, STEPSIZE)
2:     $w \leftarrow 0$
3:     for i = 1 to numIters do
4:         for x, y in examples do
5:             $w \leftarrow w - \text{stepSize} * \nabla_w \text{Loss}(x, y, w)$
       return w
```

Algorithm 2 Gradient Boosting

```
1: procedure LEARNREGPREDICTOR(EXAMPLES, NUMITERS, STEPSIZE, NUMTREES)
2:     $w \leftarrow 0$ for $i$ in range(numTrees)
3:     obj $\leftarrow y$ for $x, y$ n examples
4:     for $w_j$ in $w$ do
5:         for $i$ in range(1, numIters) do
6:             for ind in range(examples.size()) do
7:                 $w_j \leftarrow w_j - \text{stepSize} * \nabla_w \text{Loss}(x_{\text{ind}}, \text{obj}_{\text{ind}}, w_j)$
8:         for ind in range(examples.size()) do
9:             $\text{obj}_{\text{ind}} \leftarrow x_{\text{ind}} w_j - obj_{\text{ind}}$
       return $w$
```

Algorithm 3 Cross-Validation

```
1: procedure TESTPREDICTORS(DATA, NUMSPLITS) $n \leftarrow length(data)$
2:     for i in range(numSplits) do
3:         trainData ← data[1:ni/numSplits] + data[n(i+1)/numSplits:]
4:         testData ← data[ni/numSplits:n(i+1)/numSplits]
5:         Train and test Linear Regression on trainData
6:         Train and test Gradient Boosting on trainData
```

[8] https://github.com/CerJesus/CS221FinalProject