Using Machine Learning to Predict Housing Prices Given Multivariate Input

CS 221, Fall 2016

Alex Seutin and Ian Jones

-Background-

The goal of this project is to use machine learning to predict the selling prices of houses based on a number of factors. We are using data from a dataset of housing prices in King County, USA, from kaggle.com. We partitioned the dataset into a training set (19,451 data points, 90% of total dataset), and testing set (2,162 data points, 10% of total dataset).

Raw input data

For learning, our input is the following 21 features in the which make up each data point in the dataset:
id - Unique ID for each home sold
date - Date of the home sale
price - Price of each home sold
bedrooms - Number of bedrooms
bathrooms - Number of bathrooms, where .5 accounts for a room with a toilet but no shower
sqft_living - Square footage of the apartments interior living space
sqft_lot - Square footage of the land space
floors - Number of floors
waterfront - A dummy variable for whether the apartment was overlooking the waterfront or not
view - An index from 0 to 4 of how good the view of the property was
condition - An index from 1 to 5 on the condition of the apartment,
grade - An index from 1 to 13, where 1-3 falls short of building construction and design, 7 has an average level of construction and design, and 11-13 have a high quality level of construction and design.
sqft_above - The square footage of the interior housing space that is above ground level
sqft_basement - The square footage of the interior housing space that is below ground level
yr_built - The year the house was initially built
yr_renovated - The year of the house's last renovation
zipcode - What zipcode area the house is in
lat - Lattitude
long - Longitude
sqft_living15 - The square footage of interior housing living space for the nearest 15 neighbors
sqft_lot15 - The square footage of the land lots of the nearest 15 neighbors

The following is an example of a possible input, which would appear in a .csv file. These are the id, date, price, number of bedrooms, etc:
[3421079032, 20150217T000000, 221900, 3, 1, 1180, 5650, 1, 0, 0, 3, 7, 1180, 0, 1955, 0, 98178, 47.5112, -122.257, 1160, 42882]

We found that not all of these features are meaningful. For example, we immediately removed 'id' and 'date' from the input features. The unique ID for each home sold is useless information that intuitively does not contribute to the selling price of a home. We found that since the date at which each home was sold varied in our dataset only between two years, and under closer examination we found that this did not sufficiently alter the selling price of the home and therefore was not a feature that should contribute to our learning. We will explain other features that were removed/modified later.

Previous and Future Work

Since this housing data comes solely from King County, the scope of this project is relatively small - our predictor will likely not be accurate with data from other areas, however since this data is from 2014-2015, it should be accurate within King County in the current year. To extend the scope of this assignment, we can use the feature extractor we come up with on data from any other area. We will also learn the relative importance of different criteria on the price of a house - which will also extend to the greater housing market.

A similar study was done by Robin A. Dubin in 1998, in which he attempted to predict housing prices using 22 input variables. However, he only had a total dataset of 1493 observations, so he had 1000 observations for making a predictor and the remaining 493 were used for testing. Interestingly, he noted that observations could be more correlated the closer the houses were geographically located to each other. This is intuitive - houses in the same neighborhoods tend to be similarly priced - however he quantified this correlation, which is something we tried to emulate.

Other previous work includes a mobile app developed in 2015 that uses Gaussian processes to predict housing prices in London, based on five variables.

We hoped to perform better than either of these two approaches, and in the end we did.

-Introduction-

Error Quantification

Before implementing a baseline and oracle, we decided how we would quantify the accuracy of our predictions. In our error analysis, we report two values: the root mean squared error and the mean percent error.

We decided that a root mean squared error is a more intuitive error than mean squared error, because it is more representative of how many dollars our prediction is incorrect by. By taking the square root of the mean squared error, we gain more insight into the meaning of the error.

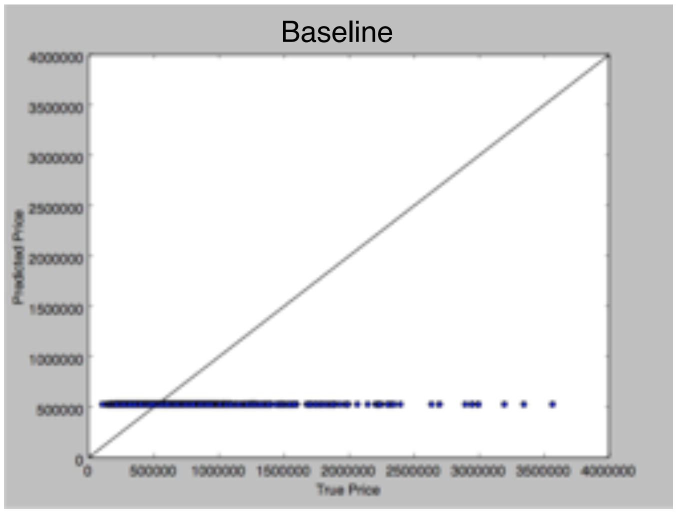
Furthermore, we report the mean percent error, which is the average percent (of the true selling price) by which predictions are incorrect.

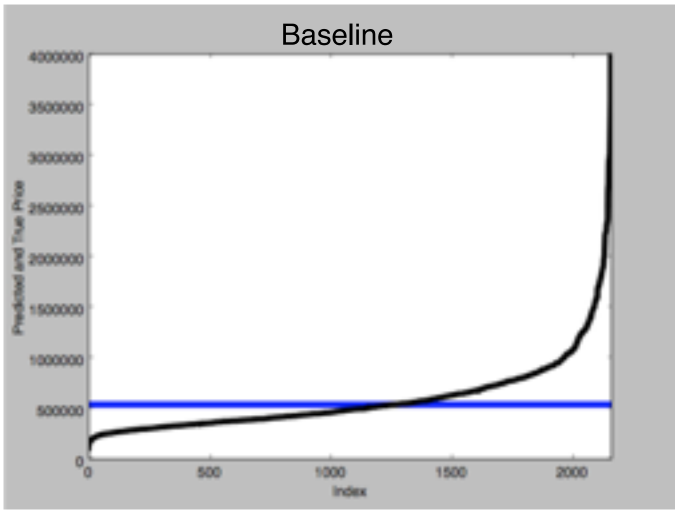
We use two types of graphs when representing the predictions and error. The first type is a graph of True Price (x) to Predicted Price (y). A perfect predictor would cause this graph to be completely linear, with a slope of 1. In the second type of graph, the True Price and the Predicted Price are both graphed against their Index in the list of testing data.

The first graph helps show how close to a linear predictor the predictor is, while the second graph more easily shows how close to the true value our predictions are.

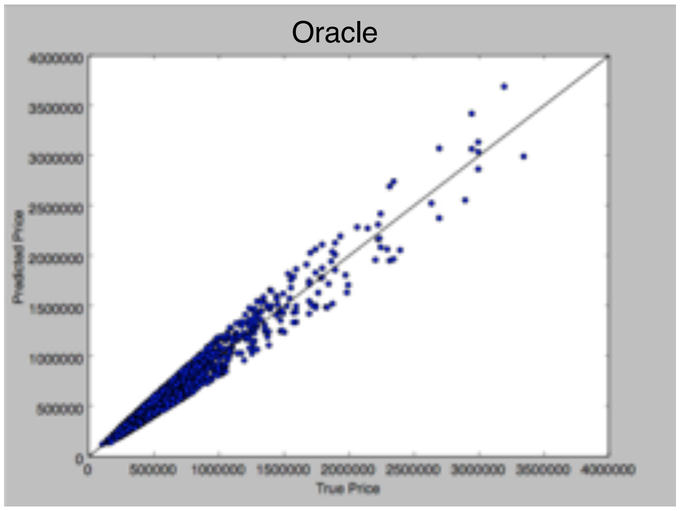
Baseline and Oracle

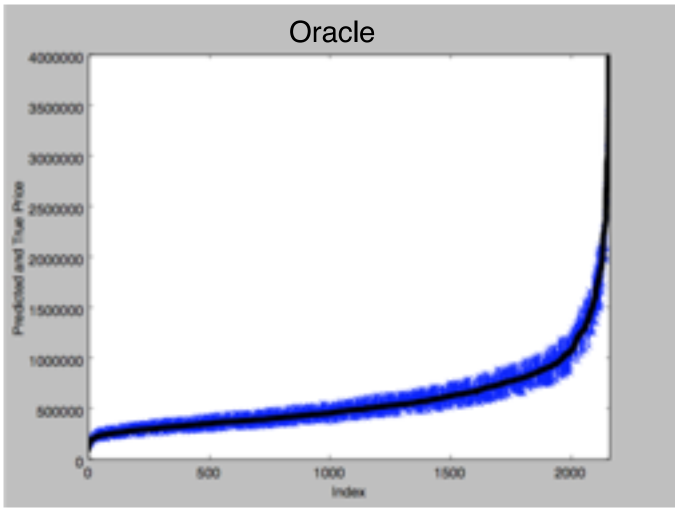
First we defined a baseline to compare our results against. We decided to define a predictor that, given any input, outputs the average selling price of a house in our training dataset as its prediction. Using the baseline as our predictor for the testing data results in a root mean squared error of 407,307.05 and a mean percent error of 40.37%.





We decided the ideal oracle would be a human real estate agent, because this person decides the listing price of the house. The real estate agent sets the listing price of the house, however in reality there is a non-negligible difference between the list price and the selling price of the house. Through our research, we've found that difference to be between 5 and 12 percent of the listing price of the house. Therefore, we'll set the upper bound of how accurately a real estate agent could possibly predict the selling price of a house for to be 95%. Since neither my partner nor I personally know a real estate agent, we decided to simulate the 5% error of a real estate agent. To do this, we used a uniform distribution to randomly predict a





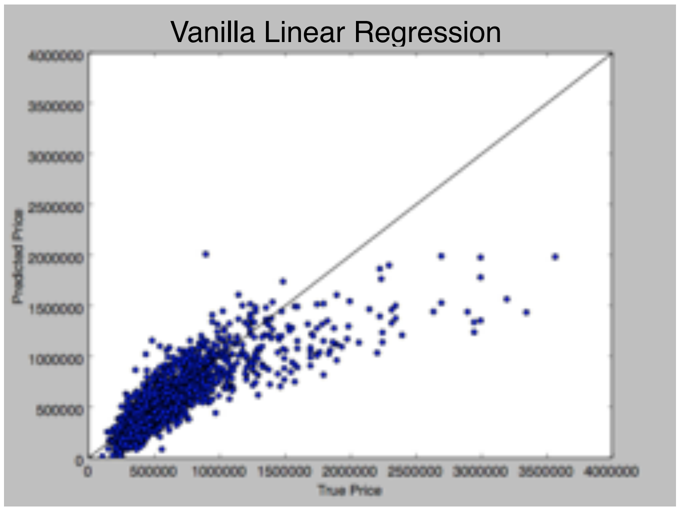
price between 90% and 110% of the selling price, given the real price. This gave us an average error of 5%. We quantified the loss using a squared loss function. We defined this as the best we would hope to achieve with our implementation of a housing price predictor.

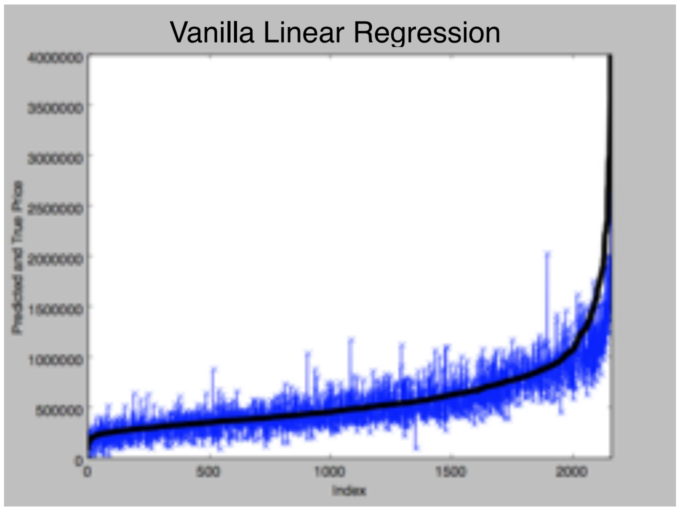
Using the oracle as our predictor for the testing data results in a root mean squared error of 85,078.13 and a mean percentage error of 10%.

-Methods-

Linear Regression

A predictor that incorporates Linear Regression seems to be the best way to model this dataset. This is because the data follow a highly linear relationship - all we have to do is select features that represent that linear relationship best. We started with a vanilla linear regression predictor, which took the raw data as input



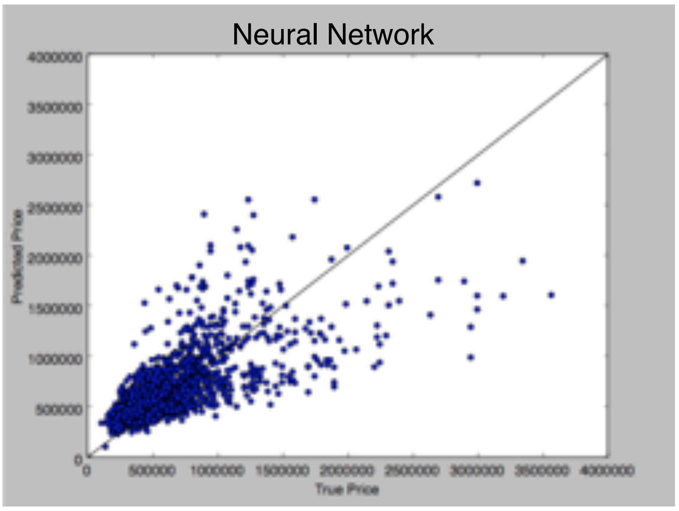


and performed no smart feature extraction. This approach resulted in a root mean squared error of 233,101.12 and a mean percentage error of 22.56%. Even this ‘lazy regressor’ performs better than the baseline, which gave us hope that linear regression had potential to perform well given well chosen features.

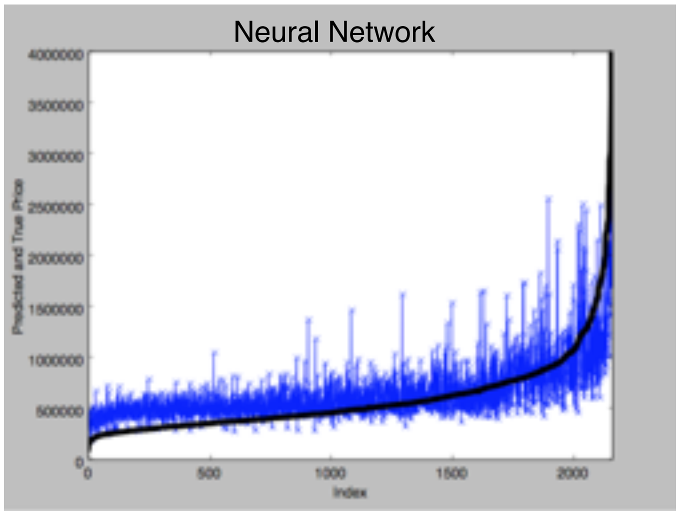
Neural Network

Next we attempted to use a neural network as our predictor, experimenting with different widths and depths for our multi-layer perception model, however we found that compared to regression we achieved worse results.

We believe that the neural net is overfitting, since using the right selection of features, this data has a relatively linear



relationship with price. We found that the neural net was overfitting to the training set, because when used it on the testing set, we achieved a higher a root mean squared error of 268,308.58 and a mean percentage error of 34.52%, which means that the neural network was outperformed by plain regression alone.



<u>Modified Beam Search</u>

Then we decided to work on a smart feature selector. We implemented what we call a 'modified the beam search' algorithm in our feature selection to create and select features that are more meaningful than the features in the raw data.

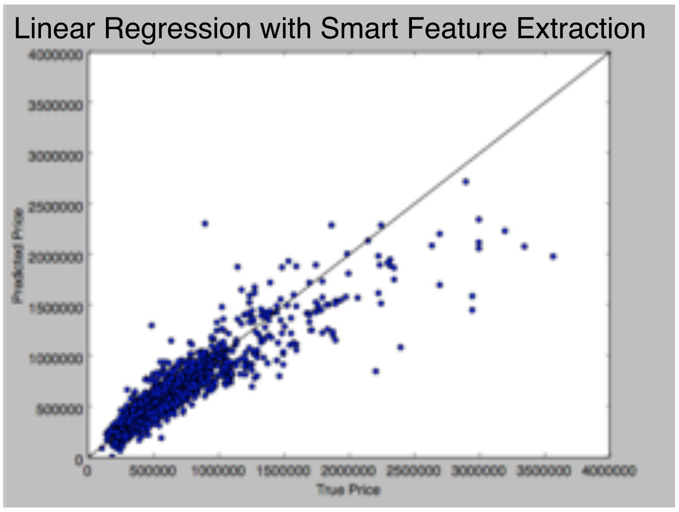
Our idea was to keep track of the K most meaningful features, extending this list by adding an order of magnitude at each step. We used automatic feature selection
sklearn.feature_selection.SelectKBest
at each step to prune. We run this algorithm T times (to a depth of T), and this results in a list of the K most meaningful features that we can then use as input for our learning algorithm.

We found that this feature selection method performed well with plain linear regression, however when we started to add and combine learning methods this feature selection started to perform worse. We think that this is because the features with magnitude 1 are the most meaningful inputs that are possible, and combining them causes overfitting to the training data.

In the end we reverted to automatic feature selection using the scikitlearn SelectKBest algorithm to reduce the number of input features.

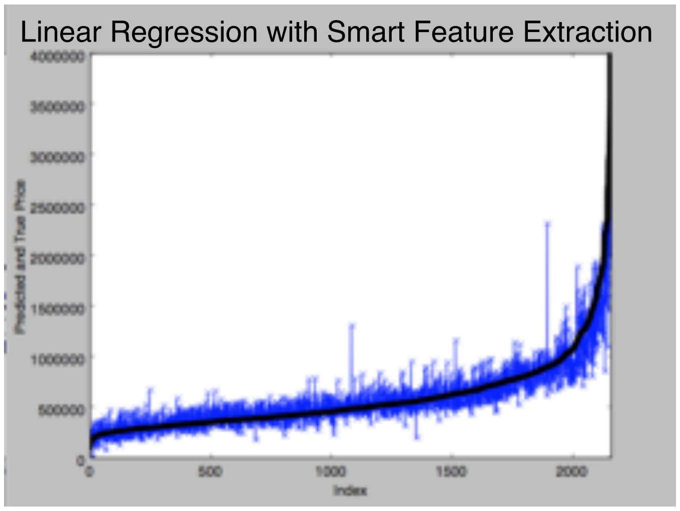
<u>Smart Feature Extraction and Removal of Extraneous Features</u>

As noted in the *Background*, we removed 'id' and 'date' from the input features, because they contain meaningless information. Furthermore, we also removed 'lat' and 'long' because using latitude and longitude as part of a linear regression model's input added extraneous information to the model. This is because latitude and longitude do not provide information that has a linear relationship of any sort.



We noticed that zip codes could be a potentially useful element of the data, because zip codes can signify what neighborhood a house is in, which has a huge influence on the selling price of the house. We noticed that there are 70 unique zip codes in Kings County, so we treated each unique zip code as a binary feature: 1 or 0 depending on if the house is located in that zip code or not. Each of these features has a separate weight when learned on. This finding alone lowered our error substantially.

We used this same strategy to create feature "bins" for 'view', 'condition', and 'grade'. The addition of these new features caused small increases in accuracy, depending on the learning model we used. These three input variables are similar to zip code in that they are classifications as opposed to values. Therefore, adding features that can recognize which "bin" the house is in according to these variables causes them to be grouped into similar groups and thus allows the predictor to be more accurate. Using the smart feature extractor and removal of extraneous features, with our linear regression model results in a root mean squared error of 182,119.70 and a mean percentage error of 15.71%.



Random Forest

The final method we tried was a random forest model from scikitlearn. We utilized the sklearn.ensemble.RandomForestRegressor, which resulted in our best results achieved by a single learning algorithm. Using the Random Forest approach with no smart feature extraction results in a mean squared error of 156,846.78 and a mean percentage error of 12.37%.

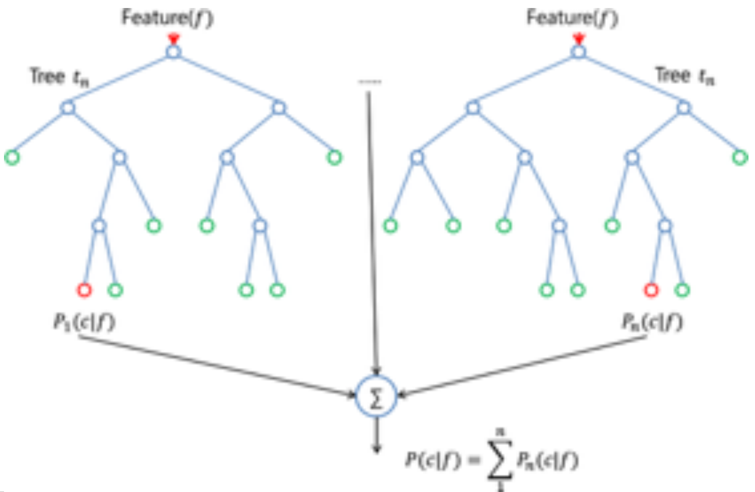
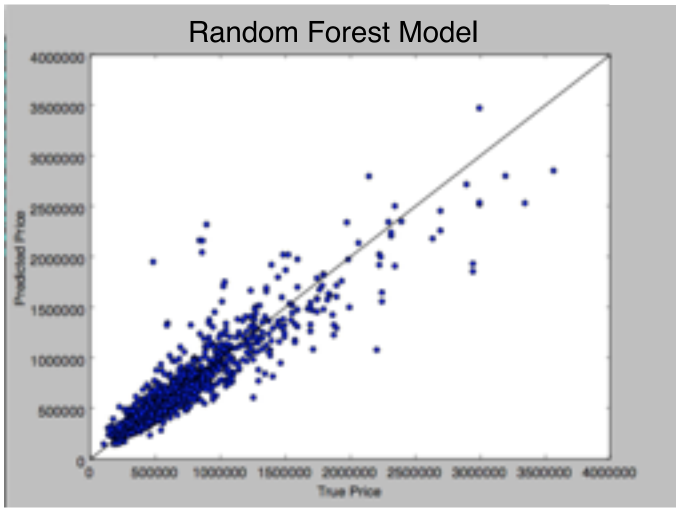
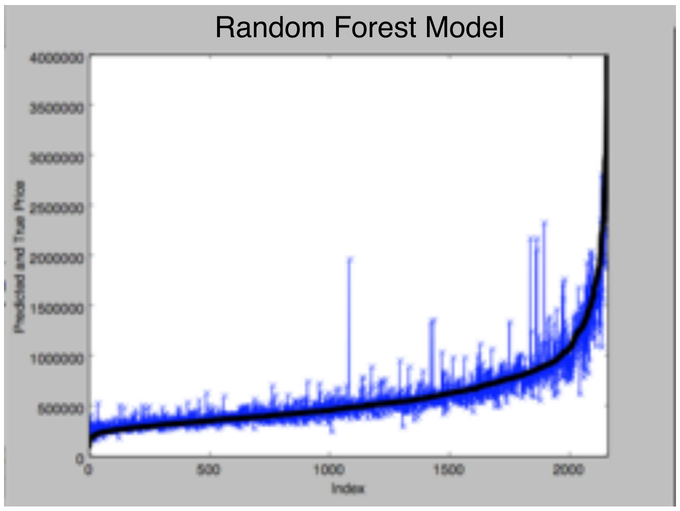


Fig. 1: Example of Random Forest

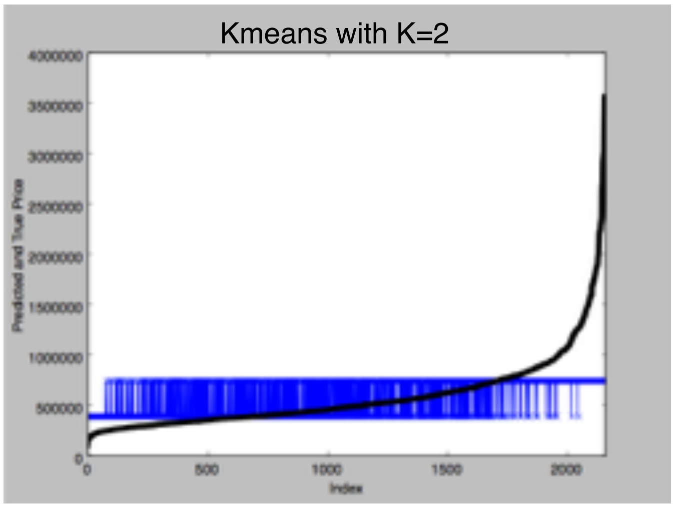


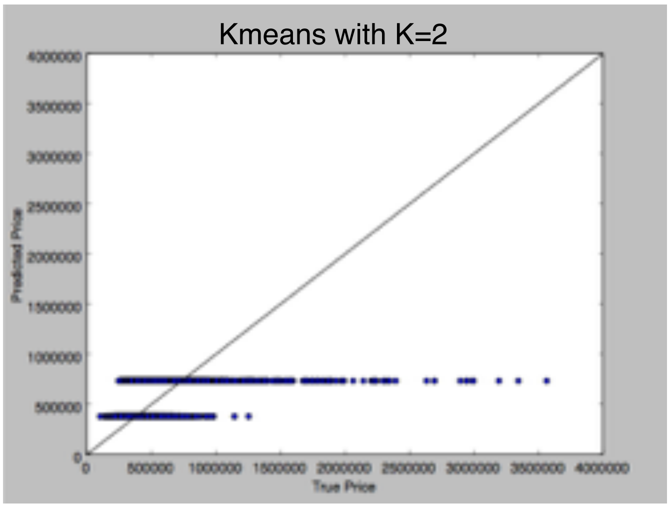


Kmeans

Next, we decided to try to partition the testing data into approximate groupings. We would do this by first sorting the training data based on price (target), then evenly dividing that data into K groups. We then created a centroid for each grouping. Next, we ran one step of k-means, assigning each x value in the training data to the nearest centroid, making sure not to ever change the actual position of the centroids we just calculated. This left us with K groups of data points that we would use to help us calculate y hat later.

We first wanted to check if this method was grouping data points into bins correctly. To do this, we implemented a new baseline approach. For each bin (comprised of training data), we would calculate the average target value for those points. We then went through the testing data and found the nearest centroid to each point. This allowed us to approximate the price range of that data point without running regression, random forest or any of other models. We then set y hat to be the average target value for that cluster - the one we just calculated using the training data. If the results were better than the baseline, then we would know that this partitioning method was doing something more than just randomly assigning points to bins. In every case, this approach surpassed the baseline. We tested this with feature extracted data

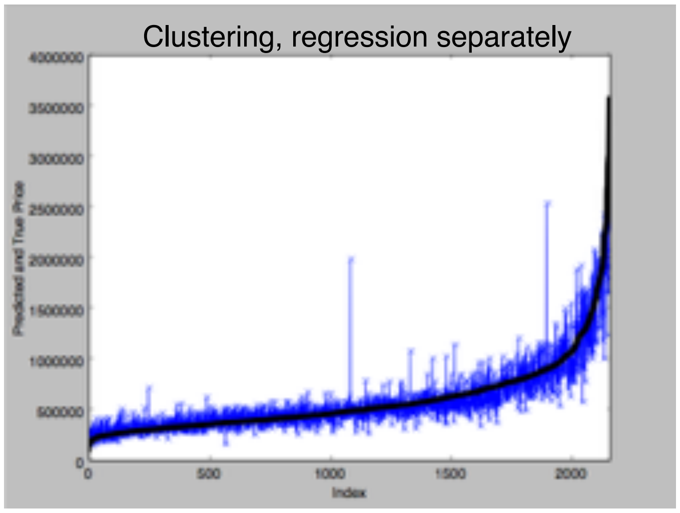


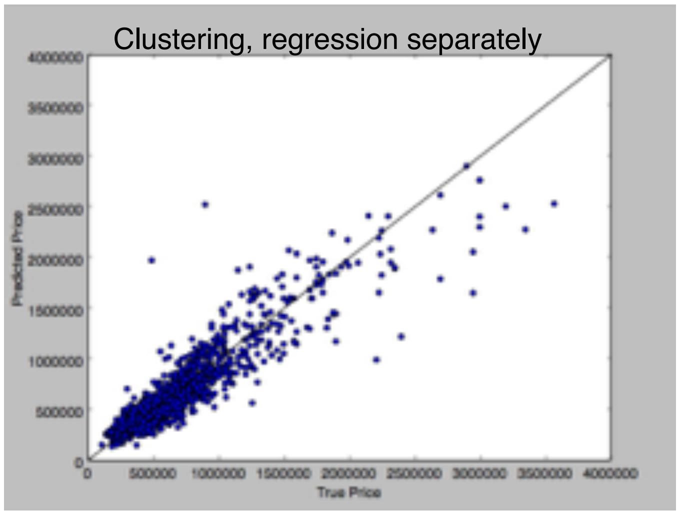


points, feature selected, beam search etc. We even tried using the plain data which would allow us to run it with a higher k value (fewer features allowed for more representative buckets with a smaller n per bucket). We found that if we used our modified beam search with our smart feature extractor to reduce the dimensionality of the centroids (leaving us with only very significant features) and set K equal to 2, we achieved a root mean squared error of 326,721.08 and a mean percentage error of 37.09%.

At one point we observed a mean percentage error of 33%. This clearly surpassed our baseline approach. With this information we could move on to more advanced approaches such as clustered regression and clustered random forests.

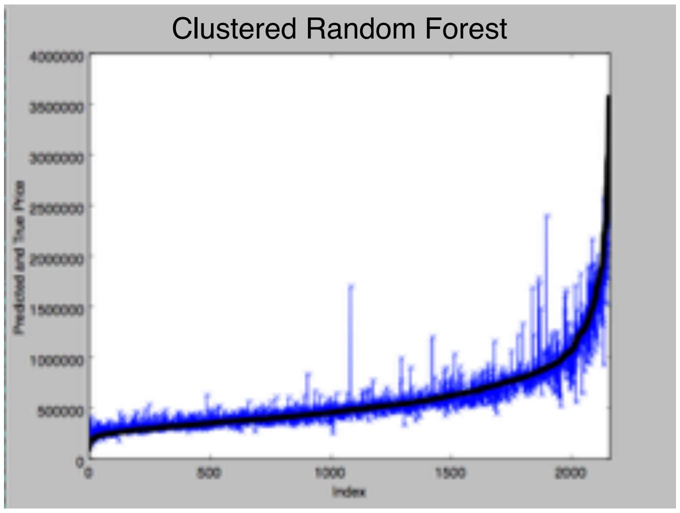
First we ran regression on each cluster separately. We used our feature extractor, allowing us to come up with the best possible regression model. Using this clustering method, we found

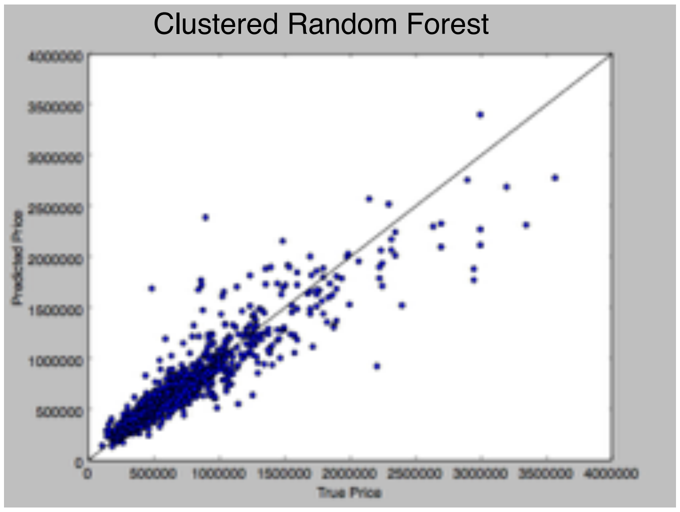




that our new results were even better than the old regression model we were using! We achieved a root mean squared error of 144,554.66 and a mean percentage error of 14.19%.

Next, we implemented a clustered random forest approach. We achieved the best results

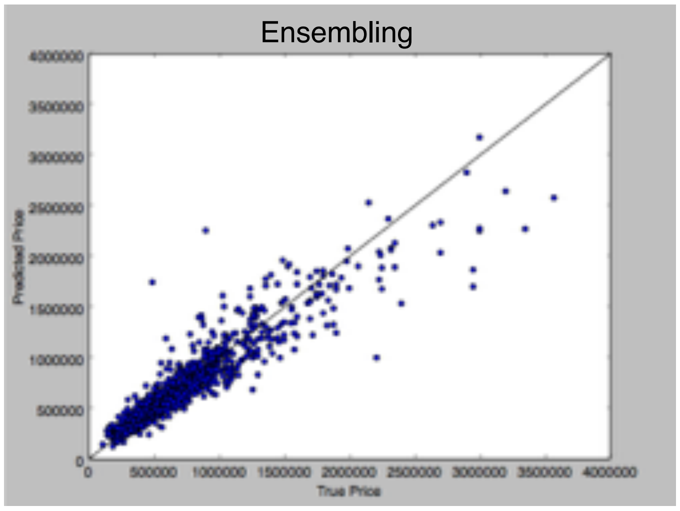


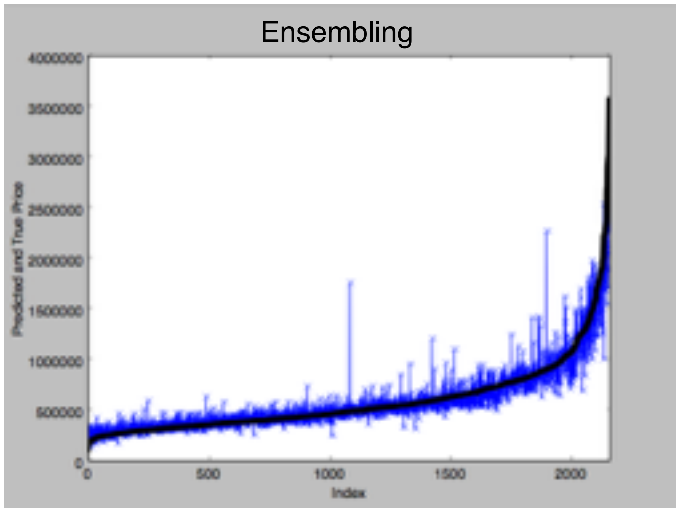


with k=2 and the number of estimators for each forest = 75. The results were incredible: a root mean squared error of 136,196.26 and a mean percentage error of 11.48%.

-Results-

Finally, we decided to ensemble our methods by taking an average over each of our top four results: regression, clustered regression, random forest, and clustered random forest. This gave us great results as well. Then, we tried using different weights for each of the methods and improved yet again so we implemented an automatic version that could test out every combination of different coefficients (for example, it would make sense that clustered random forest should be weighted higher than regression since the result was so much better). We achieved a root mean squared error of 125,590.23 and a mean percentage error of 11.07%, which far surpassed our previous best approach. This result is extremely close to our oracle and one we did not feel we could improve on any further.





-Conclusion-

This data has many limitations. The largest one is that we have no information about potential buyers. An auction can be extremely unpredictability. Ego, biding wars, buyer's susceptibility to a seller or realtors pressure, and even the weather during the time the house was on the market, are all great examples of missing information. This is why even human realtors can only predict the selling price of a home within 5-10% at best - and they have access to a lot of that missing information! Furthermore, the value of a house can be heavily influenced by features that are extremely subjective. Artistic quality of the residence, architecture, and a specific style might appeal strongly to one buyer but not another. Given a limited set of very objective, quantifiable features, we were dubious that we could come anywhere close to our oracle. However, with a combination of various approaches we were able to come up with an ensembling method that beat every single simple method we tried: by combining regression, clustered regression, random forest, and clustered random forest we actually came very close to the oracle with an average error of only about 11%.

Throughout this process we gained a far deeper understanding of the housing market as well as features that were most closely correlated with price, which could assist us in developing a more general framework for predicting housing prices in different parts of the world in the future. Finally, we expanded upon many basic methods that were introduced in lecture, gaining a very deep understanding for how they work and how we might use them for future projects.

-Works Cited-

Dubin, Robin A. "Predicting House Prices Using Multiple Listings Data." *The Journal of Real Estate Finance and Economics*. Vol 17. 1998. pg 35-59.

Kaggle: *House Sales in King County, USA*. <https://www.kaggle.com/harlfoxem/housesalesprediction>.

Scikit-learn: *Machine Learning in Python*, Pedregosa et al., JMLR 12, pp.2825-2830, 2011.