Prediction of Housing Prices of Indian Cities with AdaBoost Regression and Neural Network

Andrew Rose
Faculty of Engineering
University of Victoria
Victoria, BC V8W 2Y2
andrew.rose43@gmail.com

Hiranyeshwar Vuppala
Faculty of Engineering
University of Victoria
Victoria, BC V8W 2Y2
hvuppala@uvic.ca

Abstract

A densely-connected neural network and AdaBoost with decision trees used as base components were each used for the task of guessing housing property prices based on data from six Indian cities. These two methods and six cities were processed using two variants of the source dataset: one in which samples with missing data were dropped and one in which missing data were replaced with simple mean imputation. AdaBoost is recommended as the better of the two techniques for this purpose, but it performed poorly given imputed data.

1 Introduction

1.1 Background information

Like most sectors, the real estate sector has been transformed by technology in recent years. Nowadays, Realtor in North America employ multiple listing services (MLSes) to efficiently sift through the property market and match buyers to sellers. [1] However, MLSes are typically only made available to licensed real estate agents, so the team on this project was unable to access them. Luckily, on the other side of the world, a similar database-driven approach to the housing market and some careful web-scraping led to the existence of the dataset used in this project: detailed descriptions of tens of thousands of Indian housing properties. [2]

Given the availability of multivariate housing market data over the Internet, it would be valuable to many audiences to be able to convert such data into sale price predictions using regression. Potential applications of a machine learning tool capable of predicting housing prices from vital statistics about housing properties include:

- Use by housing sellers to obtain reasonable estimates of how much money they can expect to sell their property for.
- Use by housing buyers to determine roughly how much money they should offer for a property.
- Use by builders and developers of housing projects to gain insight into what features of housing properties contribute most to its value, and thus how to design the most desirable housing possible.
  - However, this use case could only be carried out in reality if the modeling method offers sufficient insight into the decision-making process of the regression algorithm. Since both methods used in this project are black-box methods, neither is usable for this application. Principal component analysis, for example. would be better-suited to this application, since it isolates and reveals the major factors which correlate with higher and lower value of housing properties. Exploration of this technology would be an excellent topic for future research.

Table 1: Summary of the dataset

Feature Name	Feature Description	Original Format	Transformed Format
Price	Sale price of the property	₹ (Rupees Indian currency)	Scaled between 0 and 1
Area	Area of the property	Square feet	Scaled between 0 and 1
Location	Name of neighbourhood	Text	Two features: latitude and longitude, both scaled between 0 and 1
No. of Bedrooms	-	Integers	Scaled between 0 and 1
Resale	Whether the house is new or being resold	Booleans	-
Many other boolean features; see list under table	-	Booleans	-

1.2 The dataset

The dataset contained information about thousands of properties in the Indian cities of Bangalore, Chennai, Delhi, Hyderabad, Kolkata, and Mumbai. These data comprised forty distinct features. Table 1 breaks down the contents of the original dataset and the ways in which it was processed to make it ready for training the learners. Where feasible (and where the names of the columns don't make it self-explanatory), descriptions of each of the original dataset's columns have been adapted from the Kaggle page where the original dataset is hosted. [2]

In the above table, there are many missing simple boolean features which could not be included there without consuming most of a page. They are: MaintenanceStaff, Gymnasium, SwimmingPool, LandscapedGardens, JoggingTrack, RainWaterHarvesting, IndoorGames, ShoppingMall, Intercom, SportsFacility, ATM, ClubHouse, School, 24X7Security, PowerBackup, CarParking, StaffQuarter, Cafeteria, MultipurposeRoom, Hospital, WashingMachine, Gasconnection, AC, Wifi, Children'splayarea, LiftAvailable, BED, VaastuCompliant, Microwave, GolfCourse, TV, DiningTable, Sofa, Wardrobe, and Refrigerator.

More details about the dataset can be found in the "Data munging" subsection later in this document.

1.3 Problem definition

The problem dealt with in this project was defined as creating a regression tool that could accurately guess the sale value of housing properties. Additionally, the team sought to compare their best effort at designing an optimal solution to a basic densely-connected neural network. This was intended to display how generic machine learning technology compares to a more advanced technique specially selected for this task.

1.4 Previous work

Because the Indian housing dataset is posted to Kaggle, there is no shortage of prior attempts to work with the data. One, "Housing Prices: EDA and Prediction", tests three different regression techniques on the dataset: a decision tree, a random forest, and XGBoost. It found XGBoost to be the best-performing algorithm, as evaluated using both r2 scores and mean absolute error.[4] Another project, "Metropolitan House Price", compared XGBoost, gradient boosting, and sklearn's LassoCV using mean absolute error. This project concluded that XGBoost outperformed LassoCV and gradient boosting. [5]

2 Approach

[one-sentence overview of the subsections]

2.1 Data Munging

There are some problems related to the data we choose. Bangalore had 6,207 samples, Chennai had 5014 samples, Delhi had 4998 samples, Hyderabad had 2518 samples, Kolkata had 6507 samples, and Mumbai had 7719 samples. We had 40 features for every city. The main features were the Price of the house, Area of the house, Number of bedrooms, Location, Resale, and the rest of the features were amenities such as maintenance staff, Gym, and more. However, when we used our data, we realized that we did not have complete data.

Some samples had amenities with the number 9 instead of binary, representing that amenities data was not collected. Therefore, we decided to make two data sets of each city for comparison of our results. The first data-set would consist of only samples with complete data. In the second data-set, we would perform simple mean imputation. In the first data-set after removing samples with missing data, we ended up with 1951, 2233, 2002, 2434, 75, and 1398 for Bangalore, Chennai, Delhi, Hyderabad, Kolkata, and Mumbai, respectively as shown in Figure 1.

For the location, we used the neighborhood of the location and converted it into longitude and latitude. During the conversion we concatenated neighbourhood, city and country to get accurate coordinates. For consistency of result, we removed locations whose coordinates were not available. We then normalized the whole data set so that no feature dominates the others in the neural network, which would work well with the AdaBoost regressor.

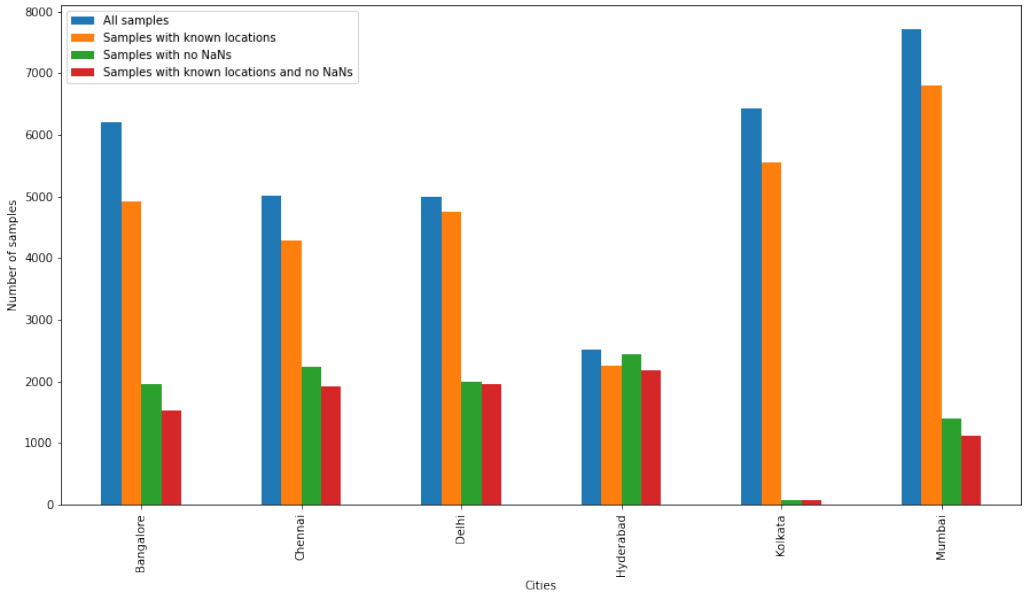

Figure 1: Graph showing how samples were removed during data munging

Figure 1 shows the number of samples left after each process. Blue shows the number of samples for each city we obtained from the Kaggle data set. For the second data set, we had the same amount of samples as total samples for simple imputation, but when we removed samples with unknown coordinates, shown in orange. Green shows the number of samples for the first data set where we removed the samples with missing data. Red shows the number of samples for the first data set where we removed the sample with missing coordinates.

2.2 AdaBoost decision tree regressor

With those algorithms explored by others, this project aimed to compare totally different algorithms. First, the team decided to use AdaBoost as the principal regressor of the project. This was because it had not been explored by any of the projects that had previously operated on this dataset, and also because the team originally supposed that categorical data such as the locations in the dataset would be handled well by a decision-tree-based algorithm. Additionally, the strong performance of boosting algorithms in other Kaggle projects using this dataset made the team curious as to whether AdaBoost would also perform well. Scikit-learn's AdaBoostRegressor class fit the bill for the team's needs, so scikit-learn tools were used both for the AdaBoost regressor and the secondary, neural-network-based regressor described in the next subsection.

However, the location data did not remain categorical. Some further research into the scikit-learn API revealed that categorical data must be converted to one-hot encoding, so the team complied - but the number of different locations within each city's data numbered in the hundreds. These hundreds of new, sparse features made it difficult for any decision tree with maximum depth 3 (as is the default setting of scikit-learn's AdaBoostRegressor) to make good use of location data, and in preliminary testing, the AdaBoostRegressor performed far worse with one-hot location data than it did with no location data at all. It required more training time, too, although this was trivial next to the far, far greater training time required by the neural-network-based regressor described in the next subsection.

This led the team to reconfigure the location data into a pair of new features: latitude and longitude. This improved the AdaboostRegressor's performance, but 1) required dropping some samples bearing locations with unknown latitude and longitude and 2) negated one of the team's original reasons for using this algorithm, since there was no longer any categorical data in the dataset.

2.3 Neural network regressor

The team also built a basic densely-connected neural network using scikit-learn's MLPRegressor class for comparison with the AdaBoost setup. The idea was to compare the regressor the team selected and optimized for this purpose with a basic, bread-and-butter machine learning algorithm.

In hindsight, using a densely-connected neural network to conduct one-dimensional regression is an obviously terrible idea. This is because there is only a single point of feedback in one-dimensional regression, as opposed to multiple points of feedback when multiple dimensions are being regressed over. This makes gradient descent extremely crude, circuitous, and slow; it's difficult to independently adjust weights within a neural network when a) the final set of weights always has its weights all adjusted by the same amount, and b) independent adjustment of other weights can only occur through the nonlinearity of the activation function.

Point b) in the above paragraph is particularly interesting, since the team's neural network used the classic ReLu activation function. ReLu was selected because it is a basic, common activation function intended to make the neural network as straightforward as possible for comparison with the AdaBoostRegressor, but its linearity above zero means that it likely hindered backpropagation in a context of single-dimensional regression. An activation function that is non-linear across its entire range (such as the sigmoid function) may have allowed the function's non-linearity to be more consistently exploited, and thus likely would have sped up independent adjustment of weights. This is, however, an untested hypothesis deserving of its own project and report.

The default number of iterations in scikit-learn's MLPRegressor class is 100, but for the above reasons, not even 5000 iterations were sufficient for the team's MLPRegressors to reliably converge. In order for training to be completed in a reasonable amount of time - a process made ten times slower by 10-fold cross-validation - the iteration limit had to be set to 500.

3 Results

Before listing the results, some final notes about the team's regressors:

- 10-fold cross-validation was used on both the AdaBoostRegressor and the MLPRegressor as a means of eliminating variability in the regressors' estimated real-world performance.

- Both regressors were optimized using squared loss, simply because that is the only available loss function in the MLPRegressor class and the two regressors were intended to be directly comparable in as many ways as possible.
- Both regressors' overall accuracy was determined by taking the mean of their R2 scores over the ten folds for each city. This was repeated for both variants of the dataset, as shown in Figures 2 and 3. To read a graph comparing R2 scores, remember that R2 scores range from negative infinity (arbitrarily bad) to 1 (perfect), and that higher values are better.

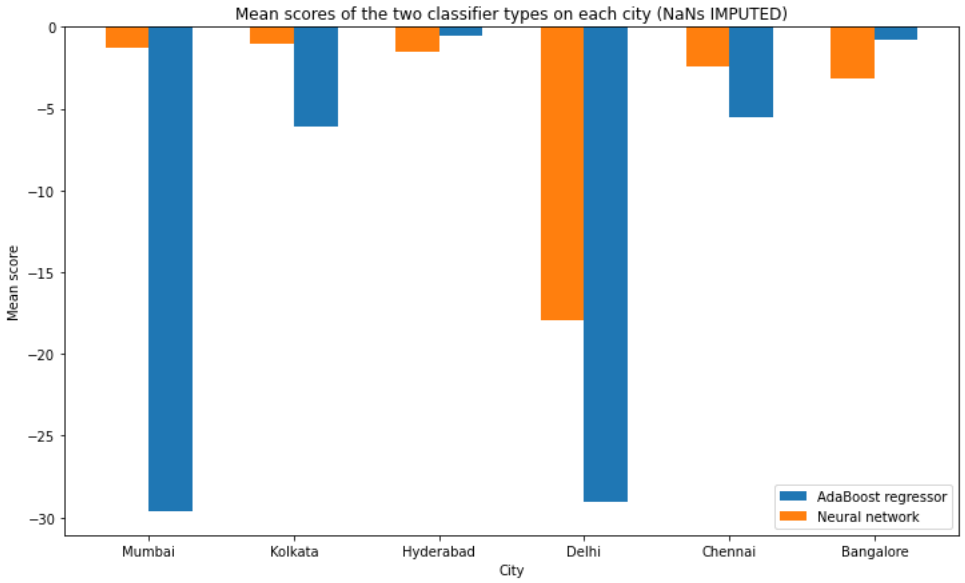

Figure 2: Graph showing the results using data with imputation

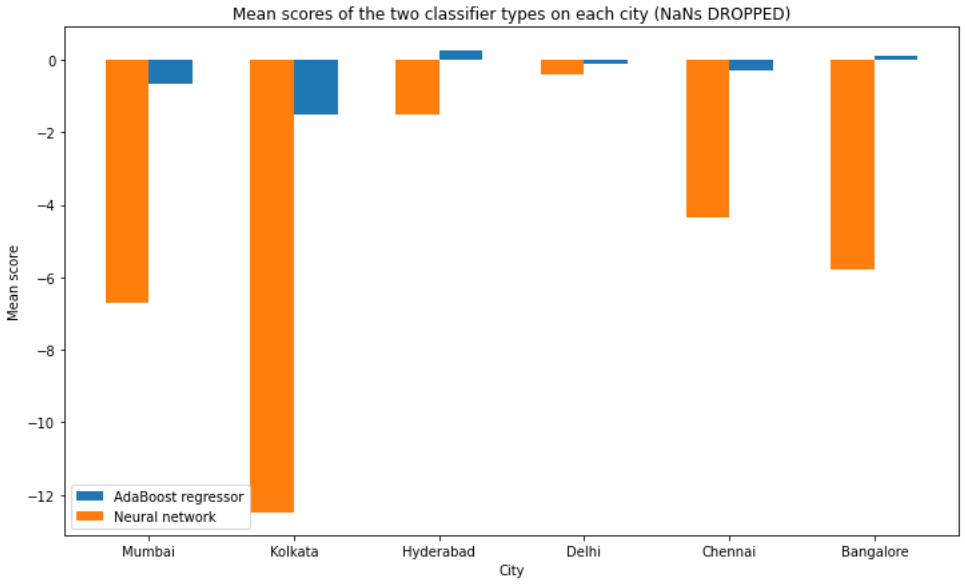

Figure 3: Graph showing the results using data where samples with missing data were dropped

At first glance, the two graphs appear to show that while the neural network usually outperformed the AdaBoostRegressor on imputed data, the AdaBoostRegressor consistently outperformed the neural

network on data in which samples with missing features were dropped. This is a misleading pair of conclusions.

Closer inspection of the graphs will reveal that the superior performance of the neural network on imputed data is still far worse than the performance of the AdaBoostRegressor on data with dropped samples. This implies that a better lesson to learn from this is that the quality of the data was seriously impaired by the imputation process. It makes sense that the decision-tree-based AdaBoostRegressor would be harmed much more by mean-based simple imputation, since its constituent decision trees would often be forced to group all those mean values on one branch. When the imputed values are binary data, as in this case, this inevitably results in the incorrect evaluation of many samples that have been grouped into the wrong branch. In contrast, a neural network that is supplied mean values allows those values to propagate through the network without causing nearly as much distortion of the output.

For these reasons, we suggest to readers of this report that while the AdaBoostRegressor can yield vastly superior results when performing one-dimensional regression, its accuracy can be ruined by mean imputation, which neural networks are not so totally derailed by. And, of course, it takes a small fraction of the training time required by a neural network in this application. The speed of training decision trees makes it more feasible to finely tune their performance and input data formatting in practice, which is no small consideration.

4 Conclusion

The team's most important conclusions are that AdaBoost regression with decision trees works well for this application and that neural networks are unsuited to one-dimensional regression (although that last point could be elaborated upon through further testing; see the topics for future work outlined below). Additionally, mean imputation appears to be highly damaging to decision-tree-based learner algorithms, whereas it has an ambiguous effect on the accuracy of neural networks.

Topics for future work may include:

- Comparison between decision trees and other base estimators within the AdaBoostRegressor setup used for this problem.

- Investigation of the effects of different imputation strategies on both decision-tree-based learners and densely-connected neural networks.

- Investigation of principal component analysis for the purpose of determining which major factors and combinations of factors determine a property's value. This could be done for the purpose of helping builders and developers of buildings to initiate projects that are in high demand, thus maximizing profit and better serving the populace's needs. Neither of the approaches in this report are suited to this task, since they are effectively black boxes.

- Experimentation with k-fold cross-validation using fewer than ten folds. This would reduce the amount of training time required by the neural networks and make it feasible to run them through more training iterations, taking them closer to convergence.

- Experimentation with different activation functions within densely-connected neural networks performing one-dimensional regression on any topic. The team hypothesizes that an activation function which is non-linear across its entire domain (such as the sigmoid function, and unlike the ReLu function used in this report) may allow a neural network in this context to converge more quickly by making it easier for weights to be independently adjusted.

References

[1] https://www.investopedia.com/terms/m/multiple-listing-service-mls.asp

[2] https://www.kaggle.com/ruchi798/housing-prices-in-metropolitan-areas-of-india

[3] https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html

[4] https://www.kaggle.com/ruchi798/housing-prices-eda-and-prediction

[5] https://www.kaggle.com/pradyut23/metropolitan-house-price