PROJECT 2 REPORT

Median House Price Predictor

Part 1

After importing the data into my Python environment, I began exploring the data. In regards to the raw data, the shape is (20634, 9), the size is 185706. This means 20634 rows and 9 columns, with 185706 total entries. It doesn't appear that any of the variables need data type conversion, so none was performed. There were no non-null values and all unique values were numerical, so no pre-processing had to be done here. There are no duplicate rows, and since all values are categorical, they don't need encoding.

I gathered some statistical information on the dataset, and I can see that while the average rooms is 5, population is 1425, and occupancy is 3, the max for each column is 141, 35682, and 1243, respectively. These are huge values and look like outliers to me, where large buildings or apartment complexes have been included in the dataset. Since the dataset is supposed to be used to predict house prices, I decided to remove these data points from the set. Here, ChatGPT helped me learn about strategies to deal with such outliers, I ended up utilizing a strategy where I capped outliers to 1.5 * the interquartile range. After capping the outliers, the extreme values were removed and the dataset was ready.

I performed univariate analysis on each of my columns and visualized them as a histogram for each column. Most columns have data distributed around the median, such as the Median Income, House Age, Latitude, and Longitude. However the rest of the columns have large outliers with sparse data, such as Average Rooms, Average Bedrooms, Population, and Average Occupancy. Such sparse data indicates that when I use data standardization techniques down the line, I should consider the MaxAbsScaler.

However, after removing the extreme values that were probably from apartment complexes, the data went to a normal distribution, which makes sense for randomly sampled values of real life data. The StandardScaler should work for this.

Part 2

The data was split into X_train, X_test, y_train, y_test using train_test_split, with a 70/30 train-test ratio, and a y stratification to maintain class proportion. I think the most important metric is the F1 score. This helped in making fair comparisons across models.

Model Performance Comparison Table

Model	Accuracy (Train)	Accuracy (Test)	Precision (Test)	Recall (Test)	F1-Score (Test)
KNN (n=3)	0.81	0.63	0.63	0.63	0.63
Decision Tree	1.00	0.83	0.83	0.83	0.83
Random Forest	0.55	0.55	0.76	0.55	0.42
AdaBoost	0.89	0.89	0.89	0.89	0.89
Histogram Based Gradient Boosting	0.96	0.91	0.91	0.91	0.91

Initially I tried KNN with n=3 and then with a grid search to again find the best n=3. The KNN model has an f1 score of 0.8 on train and 0.6 on test, meaning that it's a bit overfit on the train data, and the confusion matrix shows that it misclassified a significant number of both above and below median priced homes, showing that it can't find a clear separation between the two classes.

Then I went for a decision tree, which performed a lot better with an f1 of 1.00 on train and ~0.8 on test, however it was overfit on the training data, which is expected because it had a really deep tree which will tend to memorize the training data.

Then I went for the random forest classifier which is an ensemble that takes decision trees. This didn't overfit to the training data, but had much worse f1 and accuracy for both test data, with the confusion matrix showing that it predicted a lot of homes that were below the median as being above the median. It could probably do better with better depth tuning, which could reduce its false positive rates.

The data may have been too complex for decision trees, so I moved onto the

adaboost classifier. I utilized many different techniques with Adaboost, however all with the same base estimator as a decision tree stump and random state. I used grid search with raw data and a large search space, then I initialized a Pipeline to test out the standard scaler, MaxAbsScaler, and the RobustScaler with the same grid search space. The best performing Adaboost model was actually without scaling, achieving an F1 score of 0.89. I also tried using the robust scaler before capping outliers with the method in part 1, because this should remove the median and consider the IQR, however this preprocessing didn't result in improved performance.

The best performance resulted from using a histogram based gradient boosting algorithm that used a grid search over a large search space. It had a nearly perfect precision, recall, and f1 score for the train data, while avoiding being overfit on the train data. It performed well on the test data as well, with an f1 score being 0.91. The confusion matrix showed strong results as well, showing that it was able to generalize well to the data.

Part 3

According to the data and the analysis, I would say that the Histogram based Gradient boosting algorithm (HGB) is the best one. It was able to generalize and fit itself across all metrics, striking a good balance between accuracy, precision, and recall, as reflected in the f1 score.

I think the f1 score is the most important metric for this dataset. Since we are trying to classify homes properly as above or below the median price, its important to balance false positives and false negatives. The high precision achieved in the HGB model shows that there are many true positives and the high recall in the HGB model shows that it doesn't have many false negatives.

In conclusion, something I would like to try is utilizing different pre-processing techniques. Maybe encoding the median price from an integer to a boolean type could've helped, or if there are more advanced pre-processing techniques. I would also like to try using the deep learning approaches that we're learning in class.