April 9th, 2020 | By: David Alpert, Kayako Yamakoshi, Tejas Patel

Purchasing First Home

EGRMGMT 590.10 - New Opportunities in Big Data

Purchasing First Home

Table of Content

Introduction

To begin this entire end-to-end data science project, we spent a great deal of time trying to decide on a topic that we wanted to research. We brainstormed many topics including environmental and healthcare related problems. We agreed that rather than picking a niche topic, we wanted to find a topic that is relatable to us as well as be able to put this into our data science portfolio.

Since we are all going to be graduating from Duke in the near future and will need to find a place to live, we decided that we would work with housing data, from Kaggle (See References), so that when we are making these decisions we would be able to predict the housing price of houses based on certain features. In addition, if we were having a difficult time finding a home, we wanted to be able to input characteristics of a home and be able to identify zip-codes where such houses may be found.

Therefore, our analysis is broken down into two main parts. The first part is using a regression analysis to predict the value of houses. The dependent variable for this analysis is the price of the house and the independent variables are many different characteristics of the home, such as size, location, bedrooms, bathrooms, etc. The second part is using a classification analysis to determine what zip-code a house is likely located in based upon its features. The attributes used in this classification analysis consist of different characteristics of the home, such as size, location, bedrooms, and price.

Data Science Process

To begin the data science process, it is crucial to understand the data that is being analyzed. This should be done both visually and by looking at important summary statistics. The next step is to clean the data and prepare it for the in-depth analysis. Key parts of this include filling in missing data and dropping unnecessary columns. Next the data needs to be split into a training set that the model is built from and a test set to evaluate model performance. Finally, the data needs to be standardized and encoded so that each variable is comparable in magnitude to each other and the categorical variables can be implemented into the models.

Data Understanding

The first step that we took was to observe the dataset from a macro-level. To validate the dataset, we checked the number of instances as well as the number of variables. The dataset had 4600 instances and 18 variable columns, which gives us enough data to build models and explore different combinations of variables to use. We then took a closer look at the data to decide which variables are insightful and reliable for modeling. We checked if there is any missing value and no N/A values were observed. We also investigated the statistics to understand the maximum, minimum, and deviation of each column.

	price	bedrooms	bathrooms	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated
count	4.600000e+03	4600.000000	4600.000000	4.600000e+03	4600.000000	4600.000000	4600.000000	4600.000000	4600.000000	4600.000000	4600.000000	4600.000000
mean	5.519630e+05	3.400870	2.160815	1.485252e+04	1.512065	0.007174	0.240652	3.451739	1827.265435	312.081522	1970.786304	808.608261
std	5.638347e+05	0.908848	0.783781	3.588444e+04	0.538288	0.084404	0.778405	0.677230	862.168977	464.137228	29.731848	979.414536
min	0.000000e+00	0.000000	0.000000	6.380000e+02	1.000000	0.000000	0.000000	1.000000	370.000000	0.000000	1900.000000	0.000000
25%	3.228750e+05	3.000000	1.750000	5.000750e+03	1.000000	0.000000	0.000000	3.000000	1190.000000	0.000000	1951.000000	0.000000
50%	4.609435e+05	3.000000	2.250000	7.683000e+03	1.500000	0.000000	0.000000	3.000000	1590.000000	0.000000	1976.000000	0.000000
75%	6.549625e+05	4.000000	2.500000	1.100125e+04	2.000000	0.000000	0.000000	4.000000	2300.000000	610.000000	1997.000000	1999.000000
max	2.659000e+07	9.000000	8.000000	1.074218e+06	3.500000	1.000000	4.000000	5.000000	9410.000000	4820.000000	2014.000000	2014.000000

TABLE 1: SUMMARY STATISTICS OF THE RAW DATASET

The price ranged very widely from $0 to $27 million, which indicates that there are outliers both on the low and high end of the price data. The minimum values for the number of bedrooms and bathrooms were also 0, which shows the missing values had been replaced by 0 to fill N/A. We then created a scatterplot matrix to see distributions and correlations between variables, and noticed it was hard to find patterns as each variable was in a very different scale. Figure 1 below shows a matrix of scatterplots for the raw data, which is not standardized or encoded.

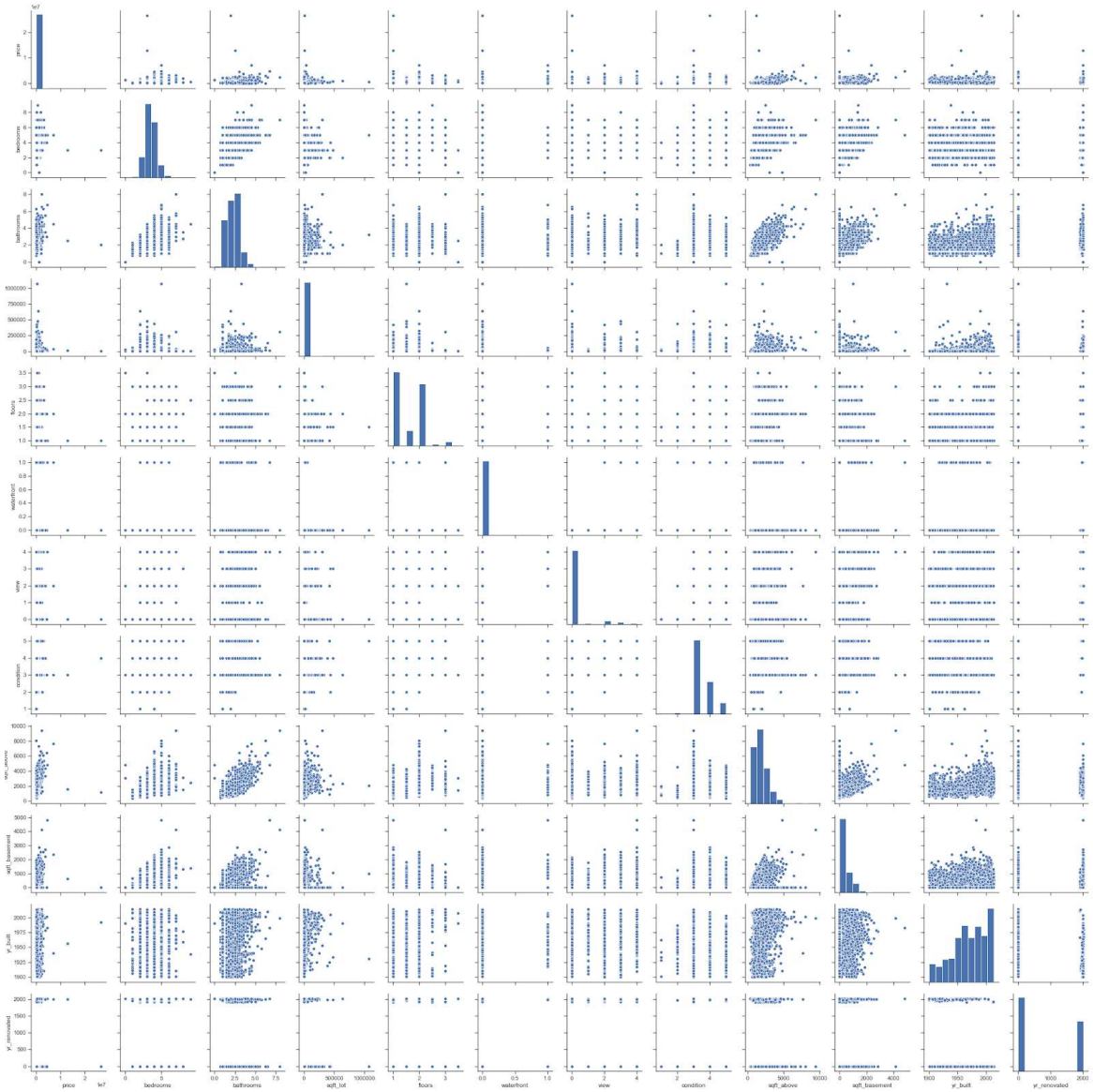


FIGURE 1: MATRIX OF SCATTERPLOTS OF THE UNTRANSFORMED RAW DATASET

Data Preparation and Cleaning

From the Data Understanding phase, we realized that we need to remove outliers, replace missing values with appropriate values, drop irrelevant variables, and transform some variables. We dropped “data-entry date”, “country”, “street”, and ”sqft-living” columns from the dataset. “Data-entry date” and “country” were not informative as all instances had the same values. We dropped “street” as it would provide similar information with “statezip” and “city” variables. “Sqft-living” column was a summation of “sqft-above” and “sqft-basement” and had to be removed so the same features would not get counted twice when building models (it is a linear combination).

As for replacing missing values with appropriate values, there were five instances that had $0 for price and one instance with price below $50,000, which seemed very low. We took the average of the housing price from all instances that had prices of over $50,000. There were also two instances that are missing values for bedrooms and bathrooms. Since both of them had housing prices of over $1 million, we took the average number of bedrooms and bathrooms in the houses that value over $1 million and replaced the missing values with it. There were also two big outliers in the dataset. We found those outliers when we were evaluating the performance of the linear regression models, which we will discuss later in this paper. We observed a high accuracy score (0.74) for the training data but a significantly lower score (0.21) for the testing data. We plotted those data and found out that training data had those two outliers, which skewed the trained model line. Therefore, we went back to the data cleaning phase and removed those two houses that are priced in excess of $20 million. We also found out that some instances have “yr_renovated” before “yr_built”, which means the house got renovated even before it was built. Those data must be false and we replaced “yr_renovated” values for them with their “yr_built” values.

Lastly, we transformed “yr_renovated” and “yr_built” columns into “yr_since_renovated” and “yr_since_built” columns. In that way, those values directly tell us how old the houses are. Since the dataset was made in 2014, we subtracted “yr_renovated” or “yr_built” from 2014 to find values for new columns.

Data Preprocessing

In the data preprocessing phase, we split the data into training data and testing data at 80%-20% ratio respectively. Although we tried to use Stratified Shuffle split initially, it has turned out this method is not suitable for the dataset since some variable classes have one instance. In other words, there were some zip-codes that only have one house in the area. Since removing those instances or adding dummy instances so that each class has more than two instances would bias the models, we decided to use a normal test train split method.

Standardization and Encoding

The last step we took before moving on to building models was to encode categorical values and standardize the data. We explored two options for encoding: OneHotEncoder and TargetEncoder. Although OneHotEncoder was a simple approach, it did not work well with the data since some classes only had one instance. TargetEncoder worked well with the data and is a more suitable approach since it maps the categorical values in a logical way and encoded values represent a probability of the target variable, which is price. Below in Figure 2 and Figure 3, are matrices of scatterplots for the transformed data. Figure 2 is for the training data and Figure 2 is for the test data.

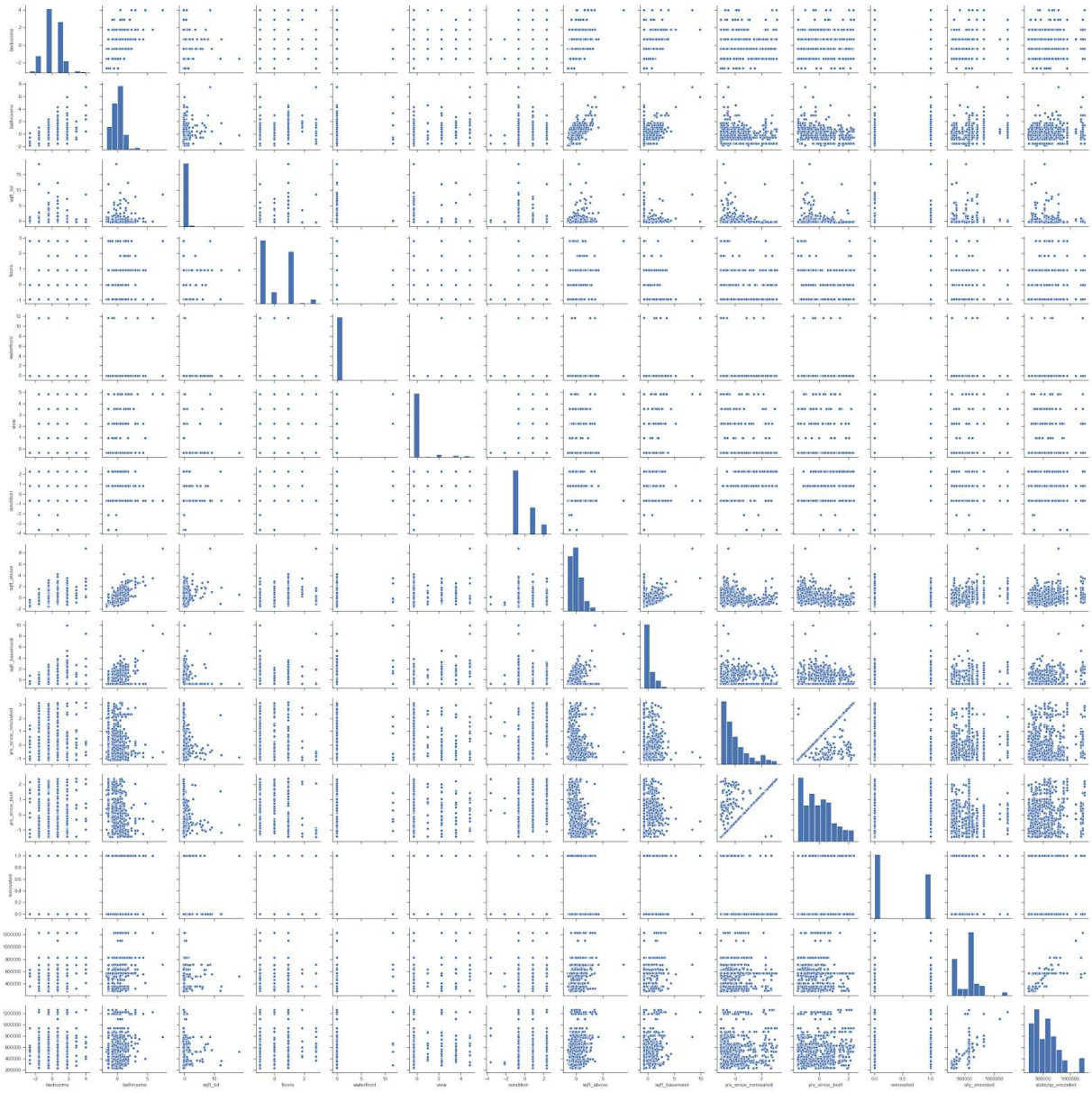


FIGURE 2: MATRIX OF SCATTERPLOTS OF THE TRANSFORMED TRAINING DATA

Data Modeling

We needed to implement a model which would truly reflect the nature of the problem statement and dataset. For the problem statement to predict the value of houses, a regression model is an ideal fit. Within the realm of regression models we wanted to focus on Linear Regression (Lasso/ Elastic), Gradient Boosting Regressor, and Random Forest Regressor. To compare the performance of these models we considered Root Mean Squared Error (RMSE) and R-Squared. Due to RMSE being sensitive to any potential outliers, we decided to bench the performance of these models on R-squared value.

To determine the zip-code of the house based upon its features we decided to implement a multiclass prediction model. We explored the options of a KNN model, Gradient Boosting Classification and a Random Forest Classifier. We decided to bench the models with an accuracy score as target variable classes in the data were well spread out among the different zip-codes and no one zip-code dominated the data set.

Regression Model

We first explored the concept of a simple linear regression, as the data complied with the four assumptions of a Linear Regression (linearity, homoscedasticity, independence, normality). We trained a Linear Regression model on the training set and checked the model performance on the test set. We used a regression model with an L1 norm Lasso optimization function. We regarded this model as our benchmark model.

Thereafter, we wanted to introduce a model validation process to the model training and perform optimization. We wanted to identify the best parameter through an optimization function. We considered: 2 optimization functions (ElasticNet and Lasso) and wanted to find the optimum number of cross-validation folds.

The introduction of the fold allowed us to significantly reduce bias and variance as most of the data is used in the validation process. We found no significant trend by varying the number of cross folds, and concluded to use 5 cross folds as the number of data points was not large and having a greater number of folds would result in overfitting the model.

Next we wanted to leverage ensemble techniques to more accurately predict the trend. We explored bagging and boosting methods, through Random Forest Regressor and Gradient Boosting Regressor. Though both models are able to contribute to accuracy, there is a common drawback of them potentially overfitting the model, and therefore we wanted to explore both models and compare its performance for our specific data set.

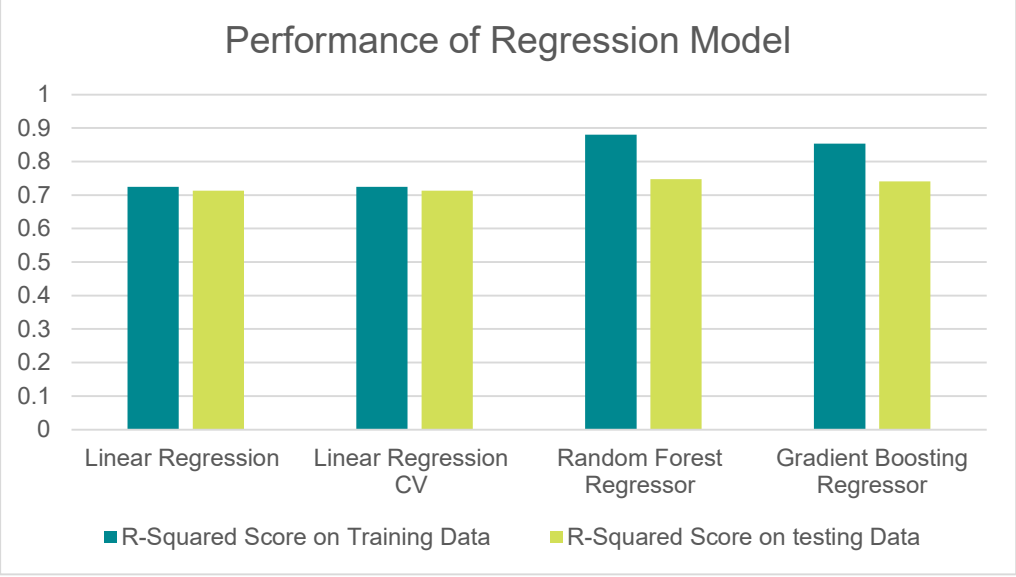
We leveraged Grid Search to tune and optimize the performance of these ensemble methods. The grid search allowed us to identify the best set of hyperparameters to implement Gradient Boosting Regressor and Random Forest Regressor. We optimized Gradient Boosting Regressor with 720 different parameter combinations and Random Forest Regressor with 800 different parameter combinations.

```
from sklearn import ensemble

# Fit GradientBoosting regression model
crossfold = [5]

params = {'n_estimators': [100,1000], 'max_depth': [2, 3, 5, 10, 15], 'min_samples_split': [2, 4, 6, 10],
          'learning_rate': [0.01,.05], 'loss': ['ls', 'lad', 'huber'], 'max_features' : ['auto', 'sqrt', 'log2']}

clf = GridSearchCV(ensemble.GradientBoostingRegressor(random_state = 42), params, cv=5, n_jobs=-1,verbose = 1)
model2 = clf.fit(Xtrain_target_Standardization,y_train)

#predictions on train data
clf_pred_train=model2.predict(Xtrain_target_Standardization)

#predictions on test data
clf_pred_test=model2.predict(Xtest_target_Standardization)

#Print and Calculate the accuracy of the model
print("r2_score on training data", r2_score(y_train, clf_pred_train))
print("r2_score on testing data", r2_score(y_test, clf_pred_test))
```

```
Fitting 5 folds for each of 720 candidates, totalling 3600 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:   54.7s
[Parallel(n_jobs=-1)]: Done 196 tasks      | elapsed:  3.1min
[Parallel(n_jobs=-1)]: Done 446 tasks      | elapsed: 11.8min
/usr/local/lib/python3.6/dist-packages/joblib/externals/loky/process_executor.py:706: UserWarning: A worker stopped w
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=-1)]: Done 796 tasks      | elapsed: 26.0min
[Parallel(n_jobs=-1)]: Done 1246 tasks      | elapsed: 114.9min
[Parallel(n_jobs=-1)]: Done 1796 tasks      | elapsed: 291.9min
[Parallel(n_jobs=-1)]: Done 2446 tasks      | elapsed: 316.9min
[Parallel(n_jobs=-1)]: Done 3196 tasks      | elapsed: 391.2min
[Parallel(n_jobs=-1)]: Done 3600 out of 3600 | elapsed: 527.0min finished
r2_score on training data 0.8531759619268248
r2_score on testing data 0.7406362296849249
```

FIGURE 3:OPTIMIZATION OF GRADIENT BOOSTING REGRESSOR

Through this process we were able to compare the different trained model performance against each other. We decided to use a Random Forest Regressor model with the following parameters as it had the overall best performance, param_grid = {'bootstrap': [True], 'max_depth': [8], 'max_features': [4], 'min_samples_leaf': [4], 'min_samples_split': [8], 'n_estimators': [1000]}. Random Forest Regressor is our chosen model as it had the latgest R-Squared Score on the Testing and Training set amongst the regression models.



Classification Model

With the limitation of time, we decided to train 3 classification models; K Nearest Neighbors Classifier, Random Forest Classifier and Gradient Boosting Classifier. We applied GridSearchCV or RandomizedSearchCV, to encapsulate elements of optimization and cross-validation into all three models.

We first explored K Nearest Neighbors Classifier, as we were confident that we had already accounted for collinearity and outliers in our data cleaning process, also we had a good understanding of the impact each of the input features has on the class selection. We optimized 'weights' and 'n_neighbors' as the hyperparameters. With the accuracy score of the K Nearest Neighbors Classifier being low, we explored ensemble methods to improve the accuracy of the model.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score


KNN = KNeighborsClassifier()

parameters = {'weights':['uniform', 'distance'], 'n_neighbors': list(range(1,50))}

cross_val = KFold(n_splits = 10)

rand_search = GridSearchCV(KNN, parameters, cv = cross_val, scoring = 'accuracy')

rand_search.fit(Xtrain_class, ytrain_class)

ypred_class = rand_search.predict(Xtest_class)

score = accuracy_score(ypred_class, ytest_class)

print(rand_search.best_params_)

print(score)
```

```
{'n_neighbors': 6, 'weights': 'distance'}
0.057608695652173914
```

FIGURE 5:OPTIMIZATION OF K NEIGHBORS CLASSIFIER

We built a Gradient Boosting Classifier, and implemented a GridSearchCV to tune 9 hyperparameters, though the model is quite accurate. Due to the number of different combinations of hyperparameters this was a time intensive process. However the accuracy scores from Gradient Boosting Classifier were significantly better.

Lastly we explored Random Forest Classifier, however due to the limitation in time. Instead of doing a full GridSearchCV, we used RandomizedSearchCV over 100 iterations. Though this is not the most extensive method to optimize. It allows it to lead to a better solution over 100 independent iterations.

Overall comparing the accuracy rates of all three models, the Random Forest Classifier performed the best. The accuracy score was 0.217, which is low. This is quite accurate as in reality there would be multiple potential zip-codes with similar probabilities since they tend to have similar houses. Being able to use further ensemble methods to provide multiple zip-codes for certain input variables is an element that we would like to implement in the future.

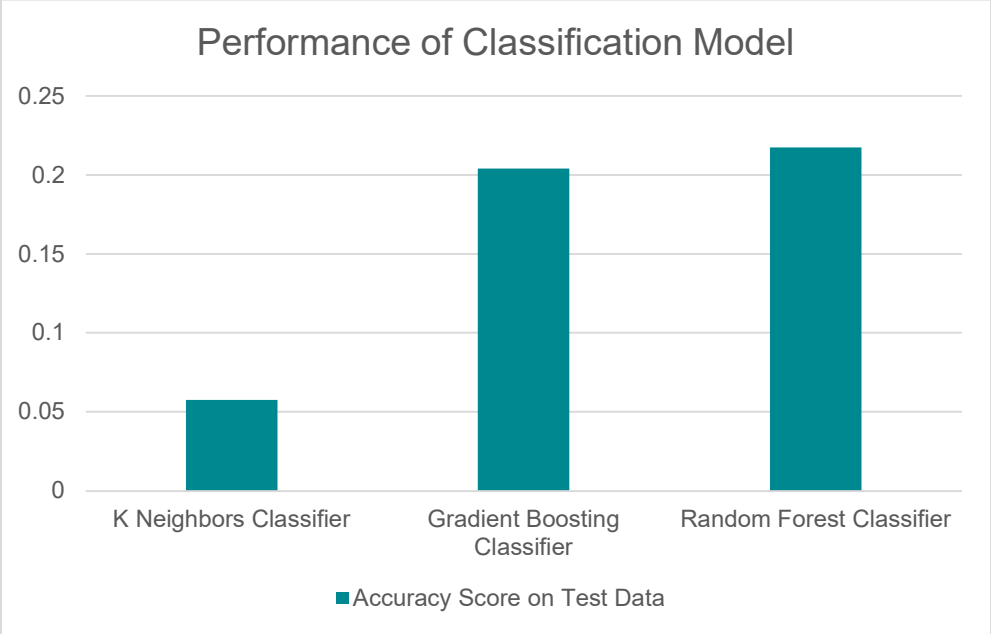


FIGURE 6: PERFORMANCE OF CLASSIFICATION MODEL

Challenges and Limitations

During our analysis, we came across multiple challenges. The first challenge that we came across was finding an appropriate dataset. Ideally, we were looking for a dataset that contained lots of data from across the entire United States so that our analysis would be applicable to all homebuyers. However, it would be too time consuming to aggregate the data on our own and validate each dataset's authenticity. To deal with this challenge, we decided to find data on one small part of the United States (Washington) and create a strong analysis regarding its housing market. Then we would be able to implement our analysis to other housing markets across the United States upon receiving new data.

The second challenge that we came across was regarding how to encode our categorical variables. We had two categorical variables in our dataset: zip-code and city. We investigated different encoders and tried to implement OneHotEncoder and TargetEncoder. When attempting to use OneHotEncoder, we faced multiple issues. For example, when we split the data into our training and test sets and then tried to encode the 77 zip-codes, some of them were only present in the test set, which caused problems. After careful consideration, we realized that using TargetEncoder for both of our categorical variables would resolve such issues and lead to a stronger analysis.

The biggest limitation of our project is related to our independent variables. In our raw dataset, there were only 16 independent variables. While these were some of the most important independent variables, such as bedrooms, bathrooms, and square footage, there are many other features that impact housing prices. For example, if a house is located near a body of water, it is likely to be more expensive than if that same house was not. If we had such independent variables, our analysis would have been much stronger.

Future Works And Research

Due to the Covid-19 pandemic this spring, the scope of our project changed quite a bit throughout the process. While we were able to address our two main questions that we set out to, there is still a lot of room for future works. Most of the future work pertains to the classification part that we were unable to complete due to the time constraints. We created three separate classification models (KNN, Gradient Boost, and Random Forest) to classify which zip-code a house is most likely to be located in based on its features. However, we were unable to take it to the next step as we had hoped to. We wanted to get the resulting probabilities from each of the classifiers and create a blending model that used the output probabilities as the inputs. This has the potential to create a stronger model that would be more accurate than any individual model.

Another potential idea for future works would be to do a more extensive search for the optimal set of hyperparameters using the GridSearchCV tool. For our models, we used the GridSearchCV, but only gave a few choices for each hyperparameter. When we tried to expand the grid to roughly 3600, it took over 8 hours to run the search. Therefore, in order to implement this, we would need a stronger computer with the ability to execute such a search.

Lastly, we believe that there may be a benefit to trying to create a clustering model. While we do not have a direct problem that leads to clustering as the best model to solve it, it may be able to provide valuable insights regarding the housing market and the different features in our dataset. A future researcher may want to create a few clustering models and feed the output into a blender, similar to how we planned to do it for classification.

Conclusion

Overall we have answered the questions that we established at the beginning of our end-to-end data science project and believe that this will be helpful to us when we search for houses in the near future. Not only did we develop the models that will predict the housing price and classify houses by zip-codes, we have applied various skills such as data cleaning, encoding, standardization, hyperparameter tuning, and cross-validation. These skills will go a long way in our data science career and to be able to apply them while learning them has given us a deeper understanding of how they work as well as their strengths and weaknesses.

References

Data Source: Kaggle
Publuished: Shree1992, 2018-08-26
https://www.kaggle.com/shree1992/housedata