**Residential House Prices Prediction in Wake County, North Carolina with Machine Learning**

Title: Residential House Prices Prediction in Wake County, North Carolina with Machine Learning

Date: Jan 24, 2024

Author: Hui Zhang

Abstract:

In this project, we explored the application of various regression algorithms to predict house prices in Wake County, North Carolina. We analyzed and preprocessed the dataset, selected relevant features, and implemented different regression models, like linear regression, random forest regression, gradient boost regression and neural networks. The performance of each model is evaluated using metrics such as R-squared, Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

**1. Introduction:**

**1.1 Background:**

The residential house market in Wake County, NC is highly competitive, with properties receiving an average of 5 offers and selling in just 37 days. To gain a competitive edge in this market, accurately assessing house prices is crucial. By leveraging historical sale data, property features and economic indicators, we aim to provide accurate and actionable pricing and remodeling recommendations to buyers and sellers.

**1.2 Objectives:**

Build a Predictive Model: Develop a machine learning model that can accurately predict residential house prices in Wake County, North Carolina.

**1.3 Datasets:**

*Historical Sale Data*: 1 csv file from Wake County, North Carolina real estate website

<https://www.wake.gov/departments-government/tax-administration/data-files-statistics-and-reports/real-estate-property-data-files>

441328 rows and 87 columns, including information about owner, mail address, real estate id, street, sale, deed, heated area, utility, exterior wall, design style, city etc.

We refer to this file as house for following discuss.

*Economic Indicators*: 1 csv file from U.S. BUREAU OF LABOR STATISTICS

<https://data.bls.gov/timeseries/LASST370000000000003?amp%253bdata_tool=XGtable&output_view=data&include_graphs=true>

51 rows and 3 columns, including population density and unemployment data from 1973 to 2023.

We refer to this file as pop\_unemploy for following discuss.

*Zipcode Longitude and Latitude*: 1 csv file from <https://gist.github.com/erichurst/7882666>

39 rows and 3 columns, including longitude and latitude data for all 39 zipcode in Wake County.

We refer to this file as zip for following discuss.

**2. Methodology:**

**2.1 Data Collection:**

download csv files from the websites listed above.

**2.2 Data Wrangling:**

Below is an overview of problems and solutions during cleaning the data:

* Problem 1: the house file is very complex and have data not only for residential houses but also has data for banks, office buildings, industrial buildings and gas stations etc.

Solution 1: select only residential houses since the objective of this project is focus on residential house prices prediction.

* Problem 2: the file has some duplicates in real estate id and location, which has same sale price, but different features.

Idea: at first, I tried to keep one row of duplicates and combine the different information, but this operation is very complex since these duplicates have too much different characteristics like exterior wall, design style, utility, heat, air etc. Just one row is very difficult to represent all the information of one group of duplicates. Finally, I decided to drop duplicates directly since the percentage of duplicates is less than 0.5%, which will not result in too much information loss. Since I only have information of street in 6 columns and some is empty string, here is a small tricky thing to get the location.

Code like this:

def concatenate(\*args):

return ' '.join(filter(lambda x: isinstance(x, str) and x.strip() != '', args))

df['location'] = df.apply(lambda x: concatenate(x[‘col1’], x[‘col2’], x[‘col3’], x[‘col4’], x[‘col5’], x[‘col6’], axis=1)

Solution 2: drop duplicates with same real estate id and then drop duplicates with same location and same zipcode.

* Problem 3: the file has lots of missing values in columns, some missing value percentage is more than 90%.

Idea: I tried to keep all these columns with missing values and label them with adding new columns, and then fill nan with 1. Code like this:

for column in df.columns:

df[column+'\_isna'] = df[column].isna()

df = df.fillna(1)

But the modeling result have no big difference and too much new features are crated since this dataset has more than 87 columns. This method maybe more useful for smaller datasets. Finally, I decided to drop columns with missing percentage more than 95% directly.

Solution 3: drop columns with missing percentage more than 95%.

* Problem 4: the file has lots of NOT formatted text or wrong datatypes.

Solution 4: remove special characters in variables and change datatypes.

Code like this:

df[column] = df[column].str.replace(',', '')

df[column] = pd.to\_numeric(df[column], errors = 'coerce')

or change from object to datetime

df[column] = pd.to\_datetime(df[column])

or change from int/float to object, vice versa.

df[column] = df[column].map({1:"string1", 2:" string2", 3:" string3"}

* Problem 5: some variables have lots of 0.

Idea: after I checked the wake county real estate website, these 0 mean not inputted data, that means they should be nan, since some has percentage more than 99%, drop then directly.

Solution 5: remove variables with percentage of 0 more than 99%.

* Problem 6: calculation of distance to nearest downtown.

Idea: the best method to calculate this is using Google Map API to get the longitude and latitude for each house, but it will be expensive for this dataset with more than 300 thousand rows, I use a simpler method, use the longitude and latitude of each zipcode for the house in this zipcode, and then calculate the distance to each city’s downtown, which is easier to get through Google Map.

Solution 6: use longitude and latitude of each zipcode as longitude and latitude for each house and then calculate the distance to each city’s downtown.

The Haversine formula for distance using longitude and latitude is:

distance = 6371\*arccos(sin(lat1)\*sin(lat2)+cos(lat1)\*cos(lat2)\*cos(long2-long1))

Note: the lat1, lat2, long1 and long2 are in radians, the result is in kms.

**2.3 Exploratory Data Analysis:**

Below is an overview of the main issues I ran into while cleaning the data:

* Figure 1: histogram of house price distribution

A graph of a house price

Description automatically generated

Findings 1: the distribution of house price is right skewed and that means it has high value outliers, which is also confirmed by the following boxplots.

* Figure 2: boxplots of house price for each city.

A graph of houses in different cities

Description automatically generatedA graph of different colored rectangular objects

Description automatically generated with medium confidence

Findings 2: most high value outliers (price more than 5 million) come from city of Raleigh. After removing these outliers, the boxplot looks much better. One interesting thing is the city New Hill, which has some low value outliers, this is different with other cities.

Note: the method to identify outliers is: 75% percentile+1.5\*IQR

* Figure 3: heatmap

A red and blue squares with white text

Description automatically generated

Findings 3: based on the heatmap, I find the house sale price have high positive correlation (correlation coefficient range from 0.46 to 0.69) with heated area, sale year, number of bathrooms and population density, this finding makes sense.

**2.4 Data Training and Modeling:**

Below is an overview of problems and solutions during data training and modeling:

* Problem 1: the dataset still has some nan value, how to handle it?

Solution 1: use fillna method with median of houses sold in same year at same city.

Code like this:

df[col] = df.groupby(['city', 'year'])['col'].transform(lambda x:x.fillna((x.median())))

* Problem 2: how to handle outliers found during EDA?

Idea1: apply transformations to the target variable to reduce the impact of outliers. Common transformations include taking the logarithm or square root. These transformations can help make the distribution more symmetrical and stabilize variance.

Idea 2: capping to set a threshold beyond which values are set to a maximum allowable value, or flooring to set a minimum threshold.

Idea 3: try robust regression models which are less sensitive to outliers, like non-linear algorithms: random forest regression, gradient boost regression etc.

Solution 3: I tried not only the linear regression, but also the random forest regression and gradient boost regression.

* Problem 4: how to avoid overfitting?

Solution 4: use cross validation in scikit-learn, for better control of the process, I used manually cross validation, which gives me more control over the train test split process and better monitoring of model performance on each shuffle.

* Problem 5: which metrics are used for model evaluation?

Solution 5: I adopted Root Mean Squared Error (RMSE) over Mean Absolute Error (MAE) for this project because RMSE gives higher weight to large errors compared to MAE. Since this dataset have high value outliers, RMSE is suitable for model evaluations to reduce the impact of large errors and improve overall accuracy. At the meantime, I also monitored the MAE and r2 score to have better understand of model performance.

Tabular Comparison:



From the above table, I can find:

exp\_1 and exp\_2 show that linear regression model works good when only use continuous variables.

exp\_4, after adding more categorical variables, linear regression model became very unstable (very large RMSE mean and variance for test set), after I check the performance for each shuffle, I found the randomness of split influence the model a lot. This is a bad sign for generalization. So, I decided to try some non-linear models.

exp\_5, random forest regression model performs much better when using all variables compared with linear regression model, but the running time is roughly 30 mins per run, so, I tried to use only continuous variables for these ensemble methods, exp\_6 is random forest regression model use only continuous variables, which has no big difference with using all variables, even better.

exp\_7, gradient boost regression model, which is the best model I tried, lowest RMSE for both train set and test set, but little bit overfitting.

exp\_8, neural network, it is surprising that it has little bit underfitting, that mean, it still has rooms for improvement, but the disadvantage is its running time is too long, 40 mins per run, maybe I should try running it on cloud GPU acceleration like Google colab.

In summary, I decide to use gradient boost regression model for hyperparameter tunning.

**2.5 Hyperparameter Tuning and Model Performance Test:**

Below is an overview of problems and solutions during hyperparameter tuning:

* Problem 1: which parameters have selected for hyperparameter tuning and how to get the best parameters?

Solution 1: I tested learning\_rate, min\_samples\_split and min\_samples\_leaf by RandomizedSearchCV in scikit-learn, to shorten the running time, I tested only 10 combinations of these parameters and use neg\_mean\_squared\_error, since the RandomizedSearchCV will try to maximize neg\_mean\_squared\_error. The best parameters in these 10 combinations are learning\_rate=0.2, min\_samples\_split=4 and min\_samples\_leaf=6. I also tried to change the max\_depth to see how it influence the model performance, and the result is listed below:



I prefer to use max\_depth=5 since it has lowest RMSE on both train set and test set, although it has little bit overfitting.

* Problem 2: what’s the most importance feature?

Solution 2: the top 10 important features are shown below:

A graph with blue bars

Description automatically generated

The above figure shows that heated area is the most important feature, and the following are population density, house sell year and house grade, which confirmed the findings from EDA.

**3. Conclusion:**

Gradient boost regression model performs best in all four models I tested.

Based on the model performance test, I find the model performs very well for house price less than 3 million, for house price higher than 3 million, the predictive ability is weak, which means the model still not very good at handling high value outliers.

**A graph with a red line and blue dots

Description automatically generated**

I also tried to use the model for recently sold house price prediction, the result is not too bad but not very good. For house ID221174, the sale price on Nov 2023 is 646000, model predicted is 568528, difference roughly 80000 dollars. For house ID445264, the sale price on Sept 2023 is 945000, model predicted is 895752, difference roughly 50000 dollars. I think the main reason for this: house price prediction is a very complex and difficult project and needs lots of accurate data especially most recent economic data and market data, the economic data used for this project is not updated and cannot reflect the real market situations.

The other thing is the data quality problem, form the below figures, I find most sample come from Raleigh, which may influence the model performance, especially for the physical city variable.

A graph of a house distribution

Description automatically generated

**4. Future Work:**

Collect more recent data into the training set could be beneficial, as it may capture evolving trends and market dynamics that impact house prices.

Explore feature engineering techniques or introducing new relevant features, such as neighborhood developments, or seasonality factors, could improve the model's ability to adapt to changing conditions.

Fine-tuning more hyperparameters of the Gradient Boosting Regressor or experimenting with alternative algorithms might also be considered to optimize predictive accuracy.

Conduct a thorough analysis of the model's residuals and identifying patterns specific to recent sales could offer insights into areas where the model may need refinement.

Combine predictions from multiple models may provide a more robust and accurate prediction for house sales.