Jiebin Zhu

IST 707

Housing Value Analysis within Staten Island

**INTRODUCTION**

The main purpose of this project is to create a model that allows me to categorize the property values of houses in Staten Island, New York based on inputting some attributes to see if that piece of property is within our budget for purchase. It also helps us to further evaluate houses that are currently on the marketplace to see if the property is either overvalued or undervalued and can allow me to be on higher “alert” when purchasing those houses. To better understand the context of this project, it is crucial to understand why I am deciding to conduct a housing value analysis on such a niche area within Staten Island, New York. During the COVID-19 Pandemic it is given that many items’ prices have gone up due to inflation by the US Government handing out stimulus checks and other benefits. In this case, property values could soon or have been affected by this macro event. Our family currently reside in Brooklyn, New York, but it is really unlikely to purchase a home that’s big and spacious enough for all of us in this borough. Therefore, we decided to look into Staten Island where houses are typically cheaper/bigger than the ones in Brooklyn. We targeted a few zip codes due to its proximity with Brooklyn and we thought about its resale value with the school zones.

The dataset was initially discovered on Kaggle (<https://www.kaggle.com/new-york-city/nyc-property-sales>), however the data was posted pre-pandemic times so I couldn’t build a model that takes into the account of the pandemic. I used the source of that Kaggle data and obtained the current data set I have from the NYC Property Sales data (<https://www1.nyc.gov/site/finance/taxes/property-rolling-sales-data.page>). I filtered the data out by its Building Class within certain zip codes my family is interested in.

**DATA**

The Data itself contains many attributes, but many aren’t useful such as the Borough, Apt number, etc. I’d like to list them over here and do a quick run through of what they are and whether if we should just get rid of it:

* **Borough** – This tells us the borough that these properties reside in Manhattan (1), Bronx (2), Brooklyn (3), Queens (4), and Staten Island (5)
* **Neighborhood** – This tells us the neighborhood area that the properties are in
* **Building Class Category** – This is a field that we are including so that users of the Rolling Sales Files can easily identify similar properties by broad usage (e.g. One Family Homes) without looking up individual Building Classes.
* **Tax Class** – Tax classes of these properties based on the use of this property:
  + Class 1: Includes most residential property of up to three units (such as one-, two-, and three-family homes and small stores or offices with one or two attached apartments), vacant land that is zoned for residential use, and most condominiums that are not more than three stories.
  + Class 2: Includes all other property that is primarily residential, such as cooperatives and condominiums.
  + Class 3: Includes property with equipment owned by a gas, telephone or electric company.
  + Class 4: Includes all other properties not included in class 1,2, and 3, such as offices, factories, warehouses, garage buildings, etc.
* **Block -** A Tax Block is a sub-division of the borough on which real properties are located.
* **Lot -** A Tax Lot is a subdivision of a Tax Block and represents the property unique location
* **Easement -** An easement is a right, such as a right of way, which allows an entity to make limited use of another’s real property. For example: MTA railroad tracks that run across a portion of another property.
* **Building Class at Present -** The Building Classification is used to describe a property’s constructive use: <https://www1.nyc.gov/assets/finance/jump/hlpbldgcode.html>
* **Address –** Address of the properties
* **Apartment Number** – If the property has an apartment number then this is included
* **Zipcode –** The postal code of the property
* **Residential Units** – Number of residential units at the listed property
* **Commercial Units** – Number of commercial units at the listed property
* **Total Units** – The total number of units at the listed property
* **Land Square Feet** – The land area of the property listed in square feet.
* **Gross Square Feet** – The total area of all the floors of a building as measured from the exterior surfaces of the outside walls of the building, including the land area and space within any building or structure on the property.
* **Year Built** – Year the structure on the property was built
* **Tax Class at Time of Sale** – Same definition as the Tax Class, but just at the time of sale
* **Building Class at time of sale –** Same definition as the Building Class, but just at the time of sale
* **Sale Price** – The price that was paid for the property (a $10 Sales Price indicates that there was a transfer of ownership without a cash consideration. There can be a number of reasons for a $10 sale including transfers of ownership from parents to children.)
* **Sale Date** – Date the property was sold

**DATA PREPROCESSING**

The main thing this project is going to look at will be the Sale Price which is provided as numbers. However, since classification algorithms will be used here for this project that will have be discretized. There are many attributes in the dataset, but there are some that will just generate extra noise within the data and is not beneficial to the model at all. Before moving on so I will have to take out those attributes.

Attributes to remove:

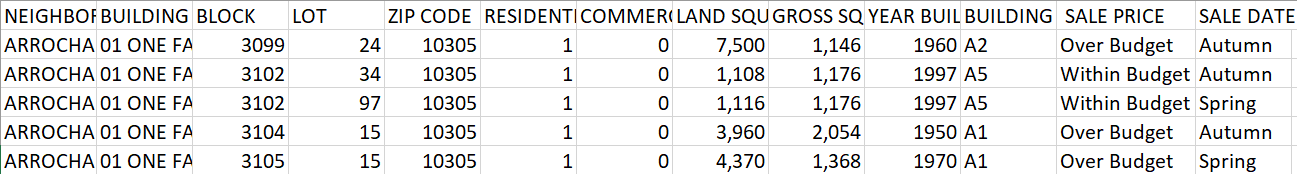
* **Borough** – The dataset that I parsed out are only for properties in Staten Island
* **Tax Class** – All the tax class within the dataset is the same, and our family wouldn’t be looking into house that have a different tax class anyway so this information is unnecessary here.
* **Easement** – Not likely going to happen since there’s only 1 example of this in the dataset. Our family would not want to have properties that have Easement.
* **Building Class at Present –** This is likely how the property was recorded before the sale will have happened, so it’s more accurate to look at the property at the time of sale since it’s often the more updated version of this
* **Address** – This is just a part of the identification for these properties and aren’t needed
* **Apartment Number -** This is just a part of the identification for these properties and aren’t needed
* **Total Units** – This is obtained from adding the residential and commercial units so I would just remove it from here
* **Tax Class at Time of Sale** – This information likely won’t change and is duplicated

What happened after removing those attributes:

1. Removed any rows that had missing data, in this case there were none
2. Removed the commas in Land Square Feet and Gross Square Feet
3. Converted the Sales Date into 4 seasons since sometimes the property value can be affected by seasonality. Months 12-2(Winter), 3-5(Spring), 6-8(Summer), 9-11(Autumn)
4. Removed all the rows that had a Sale Price of $10 since that indicates a transfer of ownership w/o payment.
5. Discretized the Sale Price into 2 categories: “Over Budget” and “Within Budget” by the budget our family is willing to pay up to. Anything below $650,000 will be within budget, and anything above that number will be over budget.
6. Convert every attribute to a nominal attribute except for Residential Units, Commercial Units, Land Square Feet, and Gross Square Year where those will be all numeric

This is what the distribution of Sale Price data currently looks like:

A sample of the data set will look like this:



**EXPERIMENT**

Classification will be the purpose here therefore I will be conducting several experiments with 4 different algorithms: K-Nearest Neighbors (KNN), Naïve Bayes, Random Forest and Support Vector Machine (SVM). The accuracy will be the metric that’s used to identify which algorithm performed the best out of the other two. The variable that I am classifying is Sale Price: “Within Budget” or “Over Budget”. I predict that SVM will have a higher accuracy rate here than the other two because of how robust it is.

For algorithms that require a random number generation, I will be using a seed of 1 for every experiment to ensure consistency for comparison purposes.

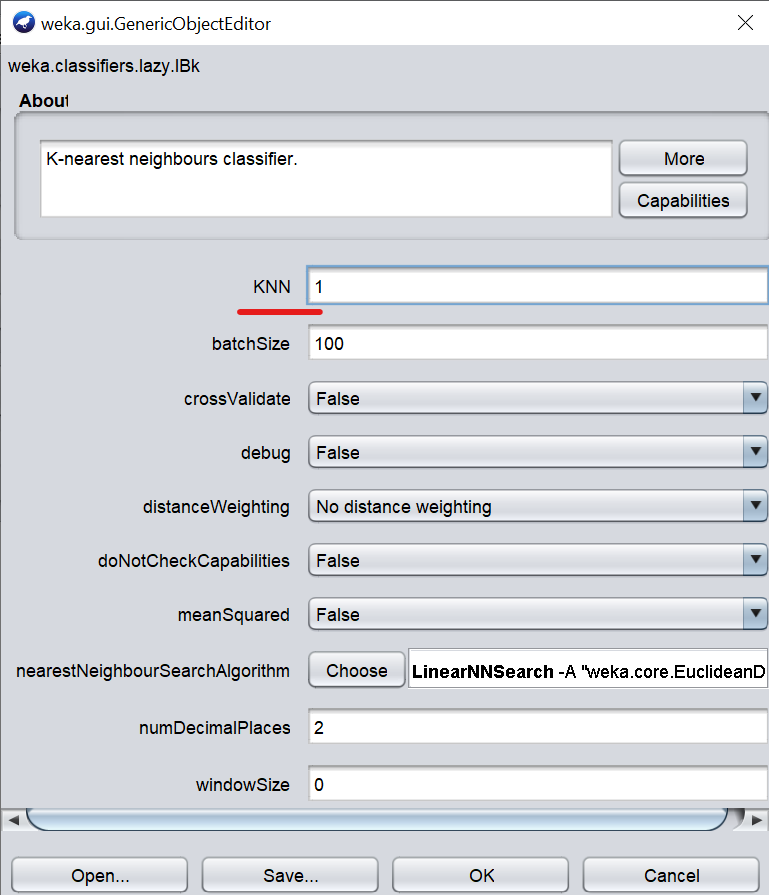
Weka will be used to conduct all the experiments on the different algorithms to provide a better visual understanding of which and what parameters are affecting the model’s accuracy rate.

As a good baseline measure I will run the default parameters on each algorithm to serve as a baseline for comparison.

Each test option for the model will be using a Cross-Validation with a 3 Fold validation.

***K-Nearest Neighbors (KNN)***

The follow parameters are the default parameters within Weka for KNN. The underlined portion is the main parameter I will be tuning to obtain the highest accuracy rate for this model, since I will be using a Euclidean Distance for this algorithm and that’s part of its default option.

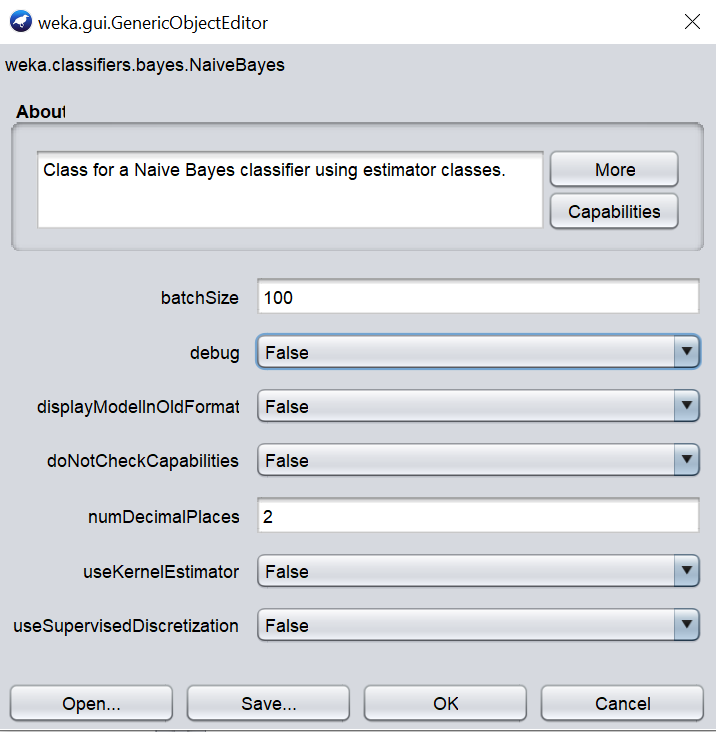


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Default KNN=1 | Exp. # 1  KNN=7 | Exp. # 2  KNN=13 | Exp. # 3  KNN=15 | Exp. # 4  KNN=17 |
| Correctly Classified Instances | 78.75% | 78.62% | 79.00% | 79.39% | 78.23% |
| Incorrectly Classified Instances | 21.25% | 21.28% | 21.00% | 20.61% | 21.77% |
| Kappa Statistic | 0.532 | 0.5288 | 0.5339 | 0.5428 | 0.5169 |
| Mean Absolute Error | 0.2529 | 0.3868 | 0.2827 | 0.2956 | 0.2878 |
| Root Mean Squared Error | 0.4095 | 0.3868 | 0.3807 | 0.3789 | 0.3785 |
| Relative Absolute error | 54.96% | 59.46% | 61.44% | 64.25% | 62.54% |
| Root Relative Squared Error | 85.40% | 80.65% | 79.39% | 79.02% | 78.93% |

This algorithm performed the best when KNN=15 and after that it reached a point where the accuracy faced a drop when I performed an extra experiment to reach that conclusion. Everything within my parameters remained as the default parameter in Weka except for the KNN number for the results to be replicated in the future. Having more “neighbors” will just generate more noise.

***Naïve Bayes***

The following parameters are the default parameters of the Naïve Bayes Classifier within Weka. There are some normal distribution within the Gross Square Feet and Land Square Feet attributes so it’s perfect for using this algorithm.



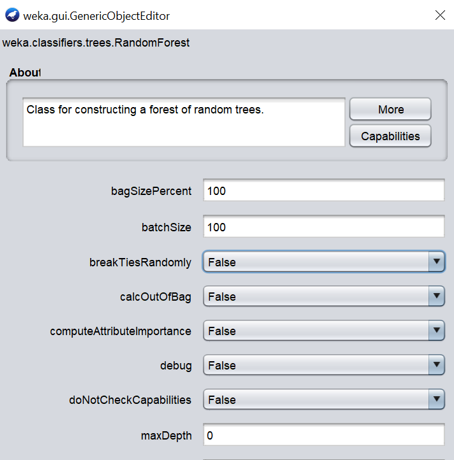
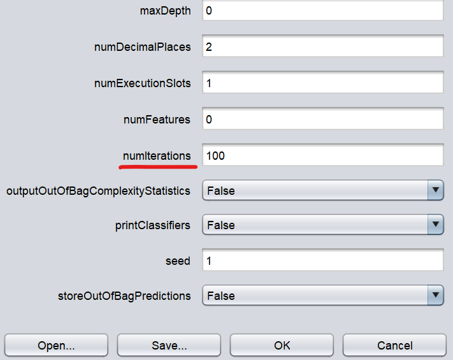
NOTE: The Experiments below will always revert to the default parameters before making the adjustment.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Default | Exp. # 1  useKernalEstimator = True | Exp. # 2  useSupervisedDiscretization = True | Exp. # 3  doNotCheckCapabilities = False |
| Correctly Classified Instances | 80.54% | 80.28% | 79.51% | 80.54% |
| Incorrectly Classified Instances | 19.46% | 19.72% | 20.49% | 19.46% |
| Kappa Statistic | 0.5795 | 0.576 | 0.5608 | 0.5795 |
| Mean Absolute Error | 0.2134 | 0.2144 | 0.223 | 0.2134 |
| Root Mean Squared Error | 0.3894 | 0.3855 | 0.3986 | 0.3894 |
| Relative Absolute error | 46.38% | 46.59% | 48.46% | 46.38% |
| Root Relative Squared Error | 81.19% | 80.38% | 83.11% | 81.19% |

The default inputs here have proved that it can provide the highest accuracy among testing it, since it performed the best out of the other experiments I conducted. The other experiments are just adding unnecessary noise and causing more complications, hence the lower accuracy rate.

***Random Forest***

This is like running a decision tree algorithm multiple times, but there’s some skepticism in this one due to the high count of nominal values some attributes may contain. The default parameters for this algorithm can be seen below. The main parameter that will be adjusted here is the numiterations which signifies how many decision trees that this algorithm will create and use as an average. If set to 1 then this is just the same as running a decision tree model.

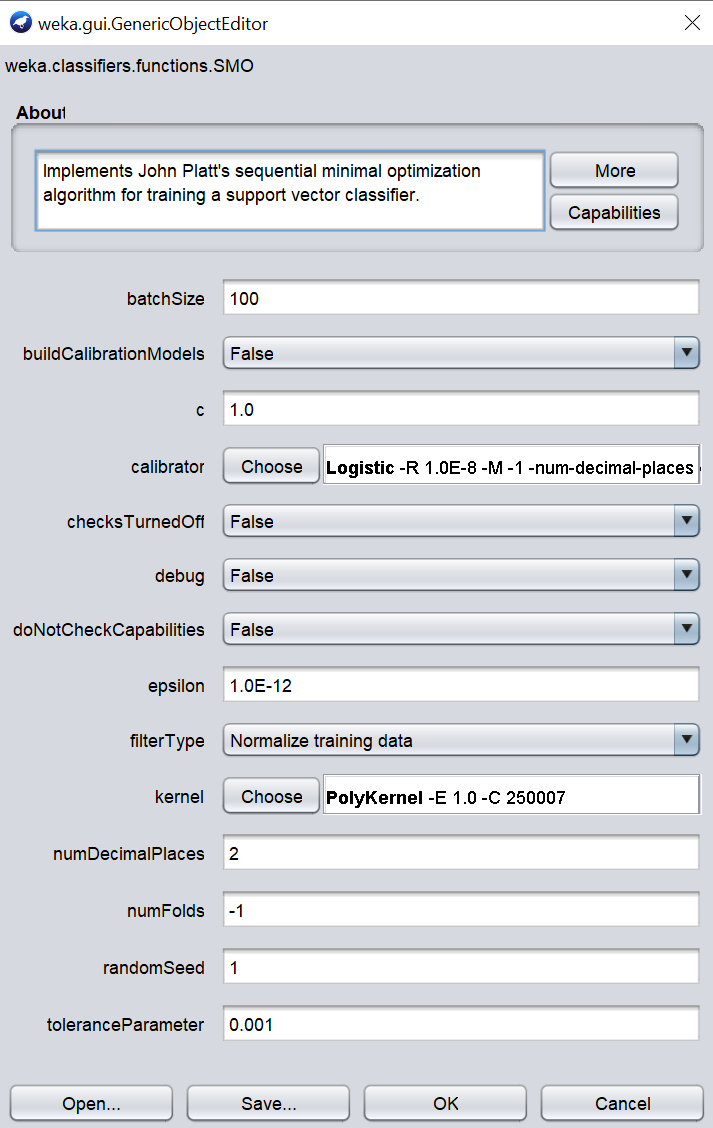
 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Default numIterations = 100 | Exp. # 1  numIterations = 50 | Exp. # 2  numIterations = 125 | Exp. # 3  numIterations = 150 |
| Correctly Classified Instances | 74.78% | 74.01% | 74.39% | 74.90% |
| Incorrectly Classified Instances | 25.22% | 25.99% | 25.61% | 25.10% |
| Kappa Statistic | 0.3917 | 0.3786 | 0.3819 | 0.3932 |
| Mean Absolute Error | 0.3651 | 0.3639 | 0.3634 | 0.3646 |
| Root Mean Squared Error | 0.4119 | 0.4136 | 0.4114 | 0.4124 |
| Relative Absolute error | 79.35% | 79.09% | 78.98% | 79.24% |
| Root Relative Squared Error | 85.89% | 86.23% | 85.79% | 85.99% |

Within this algorithm, the more trees that are being used to create a model, the more computing power and time it will take. However, in my case of having only a small amount of data and the model I am creating is for personal use, I can make my trade off and create 50 more extra trees for a bit of an accuracy boost. This is typically not the approach if the result is aiming towards scalability with larger datasets and a more company/business-oriented problem.

***Support Vector Machines (SVM)***

This is one of the most robust and useful classification algorithms for data scientists and can view data in different dimensions which could be helpful for classifying whether a house within budget or not. The default parameters for SVM is this:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Default | Exp. # 1  filterType=Standardize training data | Exp. # 2  Kernel=NormalizedPolyKernal | Exp. # 3  Kernel=NormalizedPolyKernal  buildCalibrationModels= True |
| Correctly Classified Instances | 80.67% | 74.01% | 81.18% | 80.92% |
| Incorrectly Classified Instances | 19.33% | 25.99% | 18.82% | 19.08% |
| Kappa Statistic | 0.5733 | 0.4318 | 0.5839 | 0.5823 |
| Mean Absolute Error | 0.1933 | 0.2599 | 0.1882 | 0.2326 |
| Root Mean Squared Error | 0.4397 | 0.5098 | 0.4338 | 0.3812 |
| Relative Absolute error | 42.02% | 56.49% | 40.91% | 50.56% |
| Root Relative Squared Error | 91.69% | 106.31% | 90.47% | 79.50% |

The best results were obtained when the SVM kernel was set to NormalizedPolyKernal. Having this option of normalizing the kernel helped provide a higher result. There are some attributes that looks normally distributed already, but some isn’t so that’s why having this is a major win when switching over from PolyKernel.

**MODEL COMPARISON/CONSTRAINTS**

The following table contains the best results from the 4 different algorithms that had gone through the different experiments:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | K-Nearest Neighbors (KNN) | Naïve Bayes | Random Forest | Support Vector Machines (SVM) |
| Correctly Classified Instances | 79.39% | 80.54% | 74.90% | 81.18% |
| Incorrectly Classified Instances | 20.61% | 19.46% | 25.10% | 18.82% |
| Kappa Statistic | 0.5428 | 0.5795 | 0.3932 | 0.5839 |
| Mean Absolute Error | 0.2956 | 0.2134 | 0.3646 | 0.1882 |
| Root Mean Squared Error | 0.3789 | 0.3894 | 0.4124 | 0.4338 |
| Relative Absolute error | 64.25% | 46.38% | 79.24% | 40.91% |
| Root Relative Squared Error | 79.02% | 81.19% | 85.99% | 90.47% |

From a comparison of all the different models that was built using the same data and same seed (when random number generation is implemented), SVM has the highest accuracy here. This finding was the same as the hypothesis due to how robust the model is, and it can separate the data in different dimensions to increase its accuracy. The findings here will only stay true to the data set that’s provided. The biggest constraint that this has is the amount of data that this model has. The zip codes that were analyzed had only sold about 780 residential Single or Two-Family homes in the past year, so that is the cause for the limitation.

On the scale that the models are trained, it is hard to scale it upwards for different usages. This model is solely used to target a specific type of residential properties in a certain zip codes within Staten Island. For future aspects and potential scaling of this problem to cover the entire city of New York, then more data points can be obtained, but the problem statement would also change to fit that scope. Another constraint that the current dataset has is that the discretized data is skewed towards one nominal value over another. Logically this would be the correct thing to do since that is created based on the budget that my family has, and it fits the problem statement.

**CONCLUSION**

With the power of data science, there is now a model that has been tailored towards the purpose of determining whether a house my family is looking at is within our budget or not. Another extra usage this model can do is to also to bring more “alertness” to our family when looking at house at are labeled either as “Within Budget” or “Over Budget” but it is actually the opposite on the list price. This model can only be used during the pandemic times since the economy has been heavily affected by this so the time to use it is now. Once the pandemic is over and the economy starts the recover, this model will no longer be relevant since it was built on data that was generated and impacted by the pandemic and will need new data to retrain it again if my family still has not purchased a house by that time.