Airbnb Price Prediction

Isabel Zimmerman, Drake Perkins, Tyler Livingston, Vinicius Seixas

CAP4612.01, Dr. Samarah

4 December 2019

Project Description

The overall project objective is to generate a machine learning algorithm that will evaluate a problem with high levels of accuracy. With this in mind, the group focused their efforts on the housing and tourism industry by providing predictive analytics for Airbnb rentals in New York City. Renting an Airbnb in New York City can be not only expensive, but also a hassle, since the prices vary wildly based on location and neighborhoods. This algorithm learned these price patterns and is able to make predictions on price based on these attributes.

Building the Model

We first received our data from kaggle.com, which hosts many open data sets. Our data, found here, had 16 features and nearly 50,000 instances. It was filled with many different strange characters that could not be read by python code, and had many outliers (such as houses in Manhattan that were listed for $0 a night). There was a lot of pre-processing that had to occur before we were even able to write code. Neighborhood names were transformed into numeric values, features that would add no value to the algorithm--such as apartment ID--were dropped, and strange characters were removed. This process took up the bulk of our time.

Once the data was useable, we were able to start developing our machine learning algorithm. In order to be as collaborative as possible, we used GitHub and Jupyter Notebook so all members had access to the python code. We started out with visualizing our data, in order to see a general distribution of prices. With this, we would be able to see if the results of price prediction were reasonable. In Figure 1, each Airbnb is seen as a point, graphed via latitude and longitude, with prices being on a scale of yellow (over $250 per night) to blue ($20 per night).

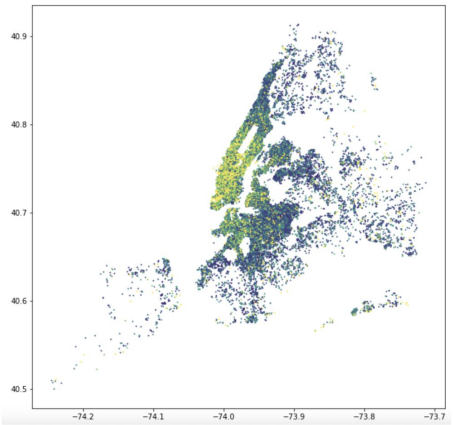

Figure 1: Map of New York City, each dot represented by an Airbnb

Next, we split the data into standard 90% training 10% test groups. We then started with linear regression, where we regressed the large neighborhoods (Manhattan, Brooklyn, etc) on price in order to see the correlation. However, this was a more simplistic model than what we

wanted to present. We contemplated k-means clustering, but decided that we wanted to continue using regressors instead of classifiers. We tested a logistic regressor and a support vector machine regressor with around 95% average accuracy, regardless of the hyperparameters, which can be seen in the appendix; at this point, the team realized we needed a more robust algorithm. We decided then to a random forest regressor; it was powerful, simple to put together, and had different versions--ie, extra trees or random patches-- to try to maximize both accuracy and efficiency. Our first trial used 100 trees. This gave us a score of 77.89% accuracy, which was close to our desired benchmark accuracy of 85%. However, with only using 6 features of the original 16, we thought the algorithm might be massively overfitting the data; that is, using 100 trees was overkill. We lowered the number of trees to 3, and set the random state to 1. After these adjustments, we had 99.64% accuracy.

Instructions for Using the System

From the Jupyter Notebook provided, “Final.ipynb”, you simply need to press run for the blocks in sequential order. Each block has comments to describe the content displayed.

The data to be used is called finaldata.csv.

Test Cases Performed

Support Vector Regressor

For our support vector regressor, we used a radial basis function kernel, since a linear model would not suffice to cover our data. The C parameter was set to 0.1, gamma 0.1, and epsilon 0.1. We had 3.825% error with this regressor. Note: although the error is shown as negative, this is expected for SVR’s “score” function.

```
In [37]: #SUPPORT VECTOR REGRESSOR
         from sklearn.svm import SVR

         X = array[1:,:6]
         Y = array[1:,6]

         X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_size=test_size, random_state=seed)

         svm = SVR(kernel='rbf', C=.1, gamma=0.1, epsilon=.1)

         _train(svm, "Support Vector", X_train, Y_train, X_test, Y_test)
         _predict(svm, "Support Vector", X_train, Y_train, X_test, Y_test)

         print("Error score:")
         svm.score(X_test,Y_test)
         #NOTE: SVM does return a negative score, -0.038, this is normal and expected

         2019-12-03 18:56:42.862210 Begin training:  Support Vector
         2019-12-03 18:57:47.944007 End training:  Support Vector
         2019-12-03 18:57:47.944327 Begin prediction:  Support Vector
         2019-12-03 18:57:56.017192 End prediction:  Support Vector
         Error score:

Out[37]: -0.038257963935729666
```

Logistic Regressor

The logistic regressor performed the worst of the group. However, it still was a strong performer with only 6.675% error.

```
In [38]: from sklearn.linear_model import LogisticRegression

         lor = LogisticRegression()
         _train(lor, "Logistic Regression", X_train, Y_train, X_test, Y_test)
         _predict(lor, "Logistic Regression", X_train, Y_train, X_test, Y_test)

         lor.score(X_test,Y_test)

         2019-12-03 19:01:56.259573 Begin training:  Logistic Regression
         2019-12-03 19:04:19.807446 End training:  Logistic Regression
         2019-12-03 19:04:19.807957 Begin prediction:  Logistic Regression
         2019-12-03 19:04:19.892566 End prediction:  Logistic Regression

Out[38]: 0.06675143038779402
```

Random Tree Regressor

We tested the feature importances, and as seen, the most important features were latitude and longitude. We also tested a single prediction for that group, and we got a price near $76.66.

Lastly, we ran the model evaluation and as seen we obtained a score of 99.68% accuracy.

```
In [10]: #How useful were the features?
         print(rnd_rgr.feature_importances_)

         [0.056557   0.30961264 0.33109043 0.1423178  0.08167924 0.07874289]

In [11]: #MAKING A SINGLE PREDICTION
         print(rnd_rgr.predict([[58, 40.72042, -73.98662, 1, 30, 200]]))

         [76.66666667]

In [12]: #EVALUATING RESULTS
         from sklearn import metrics
         from sklearn.model_selection import cross_val_score

         #score  (MSE)
         print("Accuracy: " + str(round((1-rnd_rgr.score(X_test, Y_test))*100,2)) + "%")

         Accuracy: 99.64%
```

Evaluation of the Model

Because our models used regression instead of classification, we were not using a true “accuracy score” computed from false positives, false negatives, true positives, and true negatives. Instead, our accuracy was computed by how well the line of best fit the model created could predict the prices given. In the end, the decision tree model with 3 trees was the most accurate algorithm, with 99.64% accuracy.

Known Issues and Problems

Currently the model is very sensitive to the data input. Python does not like any locations that use non-English characters, and there are occasional locations provide data way outside what is normal for other locations around them. We had to remove some of the features that were not too useful in order for the code to compile. That way, the code does not take into consideration the name of the listing, and some other features.

Lessons Learned

From this project we discovered how important data preparation is for machine learning. The data preparation phase took the longest out of all of the other phases during development. Our data had a strong effect on the final output of our algorithm as well as its complexity and processing time. We also learned that more is not better. In our Random Forest Regressor we initially used too many trees, which significantly brought our accuracy down. In the wise words of Dr. Samarah, "*Too many horses pulling the wagon pull the wagon apart."*

Future Works

If desired, this project can be extended into a marketable algorithm. It could be useful for new Airbnb users to decide an appropriate price to list their apartment, especially in big cities where prices vary from $10 to $10,000 a night in just a few square miles. If we wanted to update our data into a time-series format, we could also use historical prices to predict future housing market prices in terms of short-term rentals. This could be an algorithm used internally by Airbnb or a similar website, or it could be a forward-facing website that anyone could use.

Each Person's Contributions to the Project

Isabel Zimmerman contributed to the project by developing the SVM algorithm, collaborating to develop other algorithms, data pre-processing, taking meeting minutes, and assisted writing reports.

Vinicius contributed to the project by creating visuals and powerpoints, writing the Random Forest Regressor algorithm, some data cleaning, file compilation, etc.

Tyler created and managed the GitHub for the project. He also worked on much of the data preparation, initial models, and the expo presentation.

Drake supported the team at the Idea Expo and made contributions to the final report.

Appendix:

Link for dataset
https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data

Random Trees Regressor with lower accuracy:

```
In [26]: #RANDOM FOREST MODEL
         from sklearn.ensemble import RandomForestRegressor

         #Initialize regressor
         rnd_rgr = RandomForestRegressor(bootstrap="true", random_state=42, n_estimators=100)

         #train it
         rnd_rgr.fit(X_train, Y_train.ravel())

Out[26]: RandomForestRegressor(bootstrap='true', criterion='mse', max_depth=None,
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
                    oob_score=False, random_state=42, verbose=0, warm_start=False)

In [27]: #EVALUATING RESULTS
         print("Accuracy: " + str(round((1-rnd_rgr.score(X_test, Y_test))*100,2)) + "%")

         Accuracy: 77.43%
```

This algorithm did not perform well since it had way too many trees, which made the process inefficient as the model reproduced and made the same mistakes multiple times.