Modeling and Predicting Prices for Airbnb Listings

Hassan Hayat
Pooshan Vyas

Preface

This study comes as a result of a great number of frustration relating to exceedingly high rent prices in the San Francisco Bay Area. It is common knowledge among the people in the Bay Area that rent prices have not ceased to soar. Furthermore, the feeling of powerlessness in the face of mounting rent prices is pervasive. This feeling is substantiated as the average rental price for a single family residence in San Francisco has climbed up from $2863 in May 2012 to $4411 in May 2016. As such, any insight on these prices can help re-empower tenants and prospective tenants and inform them about the nature of these prices.

In this study, we focus on the popular online marketplace for short-term rent, Airbnb. Airbnb was founded in August 2008 in San Francisco, CA and boasts over 1,500,000 listings in 34,000 cities in 191 countries. Airbnb lists rooms and homes that are available for a short-term rent, usually, albeit not exclusively, for vacation purposes.

This study describes methods for modeling and predicting Airbnb listing prices using various Machine Learning algorithms and neighborhood-level rent estimates. This study serves the extend the similar study “Neighborhood and Price Prediction for San Francisco Airbnb Listings” by Tang, E., and Sangani, K. That original study bases its price estimates on the textual descriptions of these listings as well as on the images used to showcase the listings. We decide to move their study to another direction by instead using neighborhood-level real estate estimates to base our price estimates. As such, we posit that Airbnb prices are somewhat based on the real estate market.

Acknowledgements

We would like to take this opportunity to express our sincere gratitude to everyone who has helped us make this project successful. We would like to further thank the ridiculously expensive rent prices in San Francisco for frustrating us to become inspired to make this project.

We are also very thankful to all of the researchers and journals who have laid the theoretical foundations for this project. We also wish to particularly thank the authors of the Scikit-Learn package in Python who have provided a remarkably simple and powerful framework to use.

Last, but not least, we are deeply grateful to our project instructor and advisor, Prof. Ming-Hwa Wang, for his continued support and encouragement.

Abstract

Rent prices can be hard to understand due to the lack of information and the complexities of the real estate market. Airbnb is a platform for marketing and renting properties for short-term stays. Airbnb listings contain a lot of information that can help predict the price associated with that list. The relationship between Airbnb listing prices and average real estate prices in the vicinity of these listings is explored. Multiple Machine Learning algorithms are considered: Linear Regression, Support Vector Machines, K-Nearest Neighbor, Decision Tree, Naive Bayes, and Stochastic Gradient Descent. Finally, each of these algorithms are ranked by effectiveness and considered both with and without real estate estimates as features.

Table of Contents

Introduction

The objective of this study is to develop an elegant model for the prediction of Airbnb listing prices using Machine Learning algorithm. This study will use Airbnb listings as training and test data as well as zip code-local rent estimates from Zillow in order to improve the price prediction models.

We have studied prediction-based algorithms and different machine learning techniques which we directly use in this project. Moreover, this project will not only extend our class knowledge but also help to solve the more general real estate price prediction problem.

This study is based on the similar study “Neighborhood and Price Prediction for San Francisco Airbnb Listings” by Tang, E., and Sangani, K. In that study, the authors model Airbnb listing prices based on their textual descriptions and showcase images. We believe that we can achieve better results by considering rent estimates of the neighborhoods of each of these listings. Zillow provides average rent estimates per zip code for Single Family Residences. This research uses Single Family Residences as a baseline or representation of the overall real-estate market as it pertains to Airbnb.

Furthermore, the study by Tang and Sangani uses exclusively Support Vector Machines as their model.

In our study, we decide to use multiple algorithms:

- Linear Regression
- Support Vector Machines
- K-Nearest Neighbor
- Decision Tree
- Naive Bayes
- Stochastic Gradient Descent

The major problem is we are not aware of housing market and specific housing price, currently the prediction system is providing the housing price estimation but it's not close enough. If we want to use it as reliable decision making factor, we need to make sure our prediction is close enough to the real result. That's where scope of investigation required.

Theoretical Basis and Literature Review

The real estate market is developing rapidly and tenants, as well as realtors, have incomplete information with respect to this market. Nowadays, many websites, such as Airbnb, Zillow and others, possess great amounts of real estate data. This data can translate into valuable business insights and is amenable to prediction systems. Furthermore, this data is not fully utilized. By building a system that predicts Airbnb listing prices, we believe that we can shed light onto the more general real estate market.

Airbnb has become a symbol of the sharing economy and has changed the way people travel. As of 2015, the site advertises over 1.5 million listings in 34,000 cities around the world. In this project, we focus on listings in 16 major cities in the US, including San Francisco, where the company first started.

Airbnb listings give us a window into how participants in the sharing economy market and price their offerings. They also give us a unique insights about a city and its neighborhoods. In this study, we couple Airbnb listing data with rent price estimates and use a set of Machine Learning algorithms to develop a model that can predict the price for a given listing in a given neighborhood.

Here, we describe the Machine Learning models used in the study.

LINEAR REGRESSION

Linear Regression is the oldest and most widely used predictive model in the field of machine learning. The goal is to minimize the sum of the squared errors to fit a straight line to a set of data points.The linear regression model fits a linear function to a set of data points.

The form of the function is:

$$Y = \beta_0 + \beta_1 * X_1 + \beta_2 * X_2 + \ldots + \beta_n * X_n$$

Where Y is the target variable, and $X_1, X_2, \ldots X_n$ are the predictor variables and $\beta_1, \beta_2, \ldots \beta_n$ are the coefficients that multiply the predictor variables. β_0 is constant.

SUPPORT VECTOR MACHINES

In machine learning, support vector machines (SVMs) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

K-NEAREST NEIGHBOR

In pattern recognition, the k-Nearest Neighbors algorithm (or k-NN for short) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression.

In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

DECISION TREE

Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. It is one of the predictive modeling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a finite set of values are called classification trees. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

NAIVE BAYES

The Naive Bayes algorithm is an intuitive method that uses the probabilities of each attribute belonging to each class to make a prediction. It is the supervised learning approach you would come up with if you wanted to model a predictive modeling problem probabilistically.

Naive Bayes simplifies the calculation of probabilities by assuming that the probability of each attribute belonging to a given class value is independent of all other attributes. This is a strong assumption but results in a fast and effective method.

Given the intractable sample complexity for learning Bayesian classifiers, we must look for ways to reduce this complexity. The Naive Bayes classifier does this by making a conditional independence assumption that dramatically reduces the number of parameters to be estimated when modeling P(XlY), from our original 2(2^n –1) to just 2n.

STOCHASTIC GRADIENT DESCENT

This project will use Stochastic gradient descent algorithm, it also known as incremental gradient descent algorithms. It is very efficient approach to discriminative learning of linear classifier under convex loss function such as Support Vector machine and Logistic Regression. Stochastic gradient descent has been successfully applied to large-scale and sparse machine learning problems often encountered in Machine Learning.

The advantages of Stochastic Gradient Descent are:

- Efficiency
- Ease of implementation (lots of opportunities for code tuning)

The disadvantages of Stochastic Gradient Descent include:

- Stochastic gradient descent requires a number of hyperparameters such as the regularization parameter and the number of iterations.
- Stochastic gradient descent is sensitive to feature scaling.

The class Stochastic gradient descent Regressor in Scikit-Learn implements a plain stochastic gradient descent learning routine which supports different loss functions and penalties to fit linear regression models. Stochastic gradient descent Regressor is well suited for regression problems with a large number of training samples (> 10.000)

Hypothesis

In this study, we hypothesize that Airbnb listings prices are affected by and can be predicted with local average rent estimates. In essence, we mean that the prices listed on Airbnb for a short-term stay are related to longer-term rent prices in the same neighborhood. Given that Zillow provides average rent estimates for Single Family Residences per zip code, we have opted to use this as an Average Rent feature. For this, we assume that the rent prices for Single Family Residences represent the overall rental market.

This hypothesis can present certain challenges. Namely that not all listings on Airbnb are Single Family Residences. This simplification may introduce bias against studios and other units that do not qualify as Single Family Residences. Our hypothesis depends on the assumption that the real estate market is roughly constant in any given zip code. This is obviously untrue, but this simplification is necessary in order to make valuable predictions.

Methodology

We collect our input data from two sources: Inside Airbnb and Zillow. From Inside Airbnb, we get multiple datasets, in the form of .csv files, where each row is a different Airbnb listing. After processing the datasets, we are left with 59,932 listings, each with id, neighborhood, city, state, property type, room type, number of accommodates, number of bathrooms, number of bedrooms, number of beds, bed type and price.

From Zillow, we get a dataset containing a time series estimates for the average rent price for a Single Family Residence for most zip codes in the US. Yet, for this research, we don't need an entire time series worth of estimates. Instead, we need a single value per zip code. In order to get that single value, we consider the upload date of each Airbnb dataset and use that upload date as the canonical date for all of the listings in that dataset. For each zip code, we either extract the value at that date or, if missing, consider the closest value to that date.

In full, in order to model the "price" target feature, we consider the following features:

- Property Type (categorical)
- Room Type (categorical)
- Number of Accommodates (continuous)
- Number of Bathrooms (continuous)
- Number of Bedrooms (continuous)
- Number of Beds (continuous)
- Bed Type (categorical)
- Average Rent (continuous)

We believe that the application of machine learning algorithms on these features will yield improved results.

In order to construct our models, we have decided to use the Scikit-Learn library in Python. Scikit-Learn is a Machine Learning framework in Python containing a set of simple and efficient tools for data mining and data analysis. It is built on the popular Python frameworks NumPy and SciPy (for numeric and scientific computing) as well as Matplotlib (for visualization). Scikit-Learn is an open source framework under a BSD license. We also use Matplotlib for plotting and Pydotplus for Graph/Tree visualization.

Scikit-Learn contains a very large list of machine learning algorithms which can be easily interchanged. All of the algorithms used in this research are available as Scikit-Learn classes or methods.

The Machine Learning algorithms require two phases: a training phase and a testing phase. During the training phase, the algorithms are fed with data points containing all of the aforementioned features including the target feature. After this process, each algorithm “learns” or bases its internal constants or processes on the data it was fed. Then, during the testing phase, we feed different data points, containing all of the aforementioned features omitting the target feature, to the trained algorithms in order to produce a prediction for the target feature. We then collect all of the predictions made by all of the algorithms and compare these predictions to the actual values. Finally, we rank all of the algorithms based on how “close” their predictions are to the real values.

In order to proceed with the training and test phases, we decide to split the dataset into two parts, the training set and the test set. In order to avoid bias as much as possible, we make this selection randomly using Scikit-Learn's pre-definined shuffle-splitting function.

Furthermore, as regards the ranking of Machine Learning algorithms, we consider multiple regression metrics:

- Mean Absolute Error
- Mean Squared Error
- R^2 Error

We consider each of these metrics both separately and together in order to arrive at a more general analysis of the models.

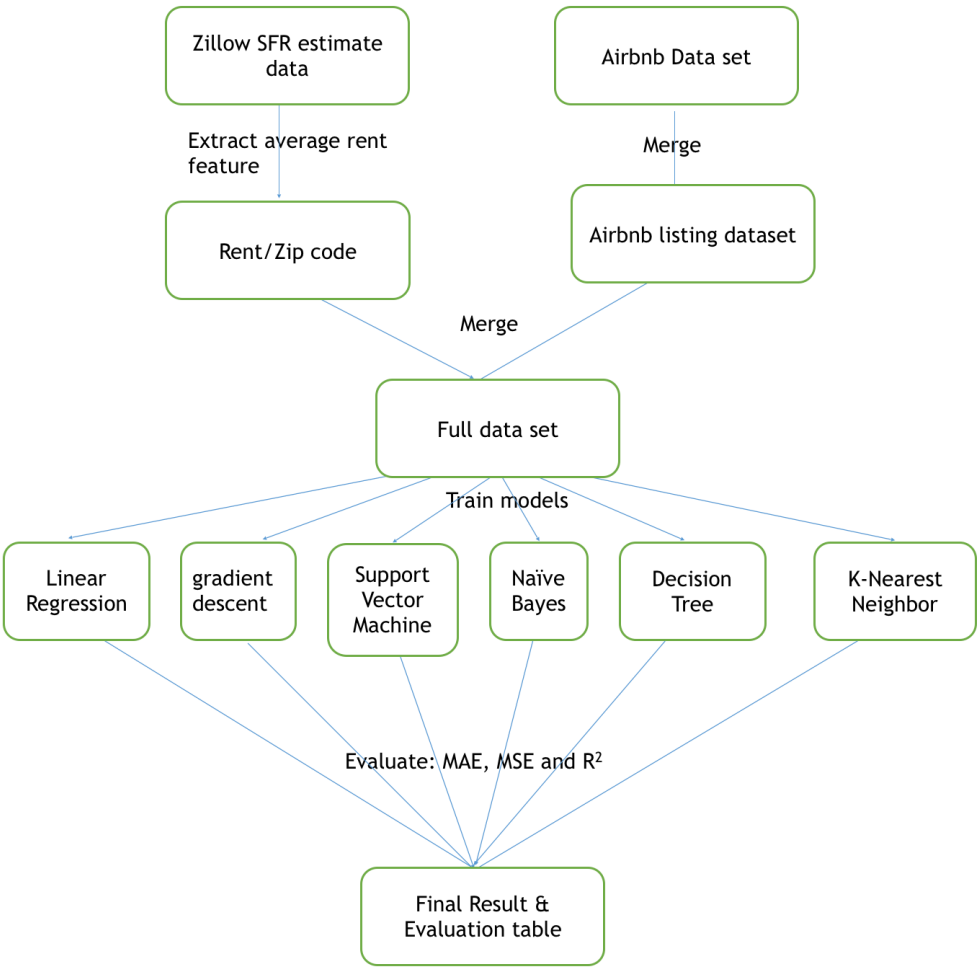
Finally, we return to our hypothesis: Local rent averages are strong indicators of Airbnb listings prices. In order to test against hypothesis, we consider our models with and without the Average Rent feature. Again, we re-apply the aforementioned regression metrics in order to compare the models with and without the Average Rent feature.

Implementation

The project has been fully implemented in Python using the popular Scikit-Learn library. Our implementation consists of three parts:

- Processing and preparing the data
- Training and storing the models and related output
- Demo

The following flowchart describes the process succinctly:



PROCESSING AND PREPARING THE DATA

In preparing and processing the data, our goal is to have 2 different datasets in the form of .csv files.

One dataset, ‘full_table.csv’, contains all the information necessary to train the models. This is a table containing every Airbnb listing in our set with 10 different columns all non-empty:

- ID
- Zipcode
- Average Rent (From Zillow)
- Property Type
- Room Type
- Accommodates
- Bathrooms
- Beds
- Bed Type
- Price

The other dataset, ‘full_zipcode_rent.csv’, contains a table mapping almost all zip codes in the US to the average rent price for a Single Family Residence in that zip code.

In order to generate these datasets, we start with a number of datasets that have to be joined together. First, we start with the datasets containing the Airbnb listings for each city/region in the set. In total, we have 16 datasets representing 16 different city/

regions, each containing multiple zip codes. These datasets had to be programmatically merged into a single dataset.

This new, merged dataset is almost ‘full_table.csv’ except that it misses the Average Rent feature from Zillow. In order to add this feature, we need to extract the data from the Zillow dataset. The Zillow dataset is a table containing the time series average rent prices for a Single Family Residence for each zip code. The problem is that the time series contains multiple monthly values for the average rent prices going from November 2010 to June 2016. In order to extract the proper value, we use the upload date of the Airbnb dataset for each dataset and use that date to extract the appropriate Zillow average rent. If a value at that is not found, we simply consider the closest value to that date. This is how ‘full_table.csv’ was generated. Again, this joining was done programmatically using a separate Python script.

Finally, to generate the ‘full_zipcode_rent.csv’ dataset, we just consider the Zillow dataset again and use the latest, or most recent, value for average rent for each zip code. Using a Python script, we can easily generate the table mapping zip codes to average rent prices.

TRAINING THE MODEL

Now that we have the data fully gathered, we can now train each model. To do this, we first go through the 'full_table.csv' dataset and gather the data into two different lists 'X' and 'Y'. 'X' is a list containing all of the descriptive features whereas 'Y' is a list containing the target features. The descriptive features are stored as a Python dictionary. Furthermore, the descriptive features are of different types—some are continuous and some are categorical—and, in order for the models to appropriately process this data, all of the data must be converted to continuous data. To do this, we use Scikit-Learn's DictVectorizer class, which converts dictionaries into lists and back and also performs a one-hot encoding of the categorical features.

After this, we use Scikit-Learn's Normalizer class to normalize the descriptive features. Normalization is only required for Gradient Descent and as such, we create a separate list called 'X_norm' that will hold the data normalized.

We then split the data into a training set and a testing set. Using Scikit-Learn's 'train_test_split' function, we have opted for a 2/3 - 1/3 split between the training and the test set.

Then, for each model, we repeat the same process. We train the model, using the 'fit' method, we then make predictions of the test features, finally we compare the predictions to the actual target values. These comparisons are in the form of the Mean Squared Error, the Mean Absolute Error, and the R2 score.

For the Decision Tree Model, we consider different models for different max-depths of the decision tree, ranging from a max-depth of 1 to a max-depth of 30. We then select the model for the list that produces the smallest Mean Squared Error. We

further decide to graph these relationships using Matplotlib. The same is done for KNN, as we consider different KNN models for different number of neighbors, ranging from a 1 neighbor to 50 neighbors. Finally, we also output the decision tree as a graph using Pydotplus.

Furthermore, all the models, as well as the vectorizer and normalizer are stored on disk as '.pkl' files that Scikit-Learn can then invoke at will. This is very practical as this means that we can store a trained model on disk and not have to re-train it overtime it is used.

After all the models are processed, we output the results of their evaluation (i.e. MSE, MAE, and R2) as a .csv file. We then repeat the same process while omitting the the Average Rent feature. This will gives us two sets of evaluated models that we then may compare to assess the usefulness of the Average Rent feature.

DEMO

The demo is a simple Python script that imports all the trained models and gets demo data from 'demo_data.csv' and uses this data to make predictions. This demo can have actual pragmatic uses as it can be used to generate a nightly rental price for a home/apartment that would like to go on Airbnb.

Data Analysis and Discussion

We consider two different evaluation outputs: evaluation with average rent feature and without.

Evaluation of Models with Average Rent Feature

Model	Mean Squared Error	Mean Absolute Error	R2 Score
KNN(#neighbors = 9)	8013.354	51.478	0.528
Decision Tree(max-depth = 7)	8236.784	53.348	0.515
Linear Regression	8926.034	57.783	0.474
Support Vector Machines	14075.838	62.934	0.171
Gradient Descent	17016.612	87.120	0.000
Naive Bayes	39730.775	135.761	-1.341

Evaluation of Models without Average Rent Feature

Model	Mean Squared Error	Mean Absolute Error	R2 Score
Decision Tree(max-depth = 5)	8710.870	56.698	0.472
KNN(#neighbors = 37)	8850.534	56.830	0.464
Support Vector Machines	9829.794	55.490	0.405
Gradient Descent	11692.555	68.772	0.300
Naive Bayes	129409.644	282.266	-6.838
Linear Regression	2205336894529920000000	438046158.443	-133573133628885000.000

Given these outputs, we can notice a number of things. First of all, in both scenarios, Decision Tree and KNN are the models that perform best. Second, we can notice an increase in the R2 scores and a decrease in the MSE and MAE for Decision Tree and KNN when the Average Rent feature is included. This therefore supports our hypothesis that the average rent prices in the area affect the price of an Airbnb listing.

As for the abnormal cases, the Airbnb data is filled with unusual listings. Some listings are claimed to be tipis whereas others are claimed to be yurts or igloos. When these individual listings are checked, it turns out that these listings are not actually what they say there are. They are usually regular apartments. This therefore means that the data contains some wrong data. Because of the one-hot encoding of categorical features, these listings with wrongs data then get their own features and cause weird results. For example, say you take a sample apartment for which you wish to predict the price, if you were to change the property type from ‘Apartment’ to ‘Tipi’, you would notice the price increase. This makes no sense as tipis are more rudimentary than apartments (tipis usually lack beds and showers). Furthermore, and most importantly, tipis don’t appear in urban areas. As such, the models, following the process of ‘Garbage in — Garbage out’, make nonsense predictions when the property type is nonsensical.

Finally, when analyzing the evaluation outputs, we can conclude that the best way to model Airbnb prices is via a KNN regression model with 9 neighbors while including the average rent prices from Zillow as a feature. This makes intuitive sense as it means that the best way to price a listing on Airbnb is to look at similar listings. This is what people do intuitively.

Conclusions and Recommendations

In short, we have seen that it is possible to model Airbnb rental prices using Machine Learning and develop models that have acceptable R2 scores. The problem is that these R2 scores are not terribly high, which attests to the volatility in the Airbnb listing market.

Furthermore, we have also shown the importance of local average rent prices in modeling Airbnb listing prices. This attests to the validity of the old real estate mantra of 'Location, location, location'. The price of a unit depends on where it is situated. Hence, if a unit is situated in an area where rent prices are high, the rental price of the unit will be higher.

Yet, we are slightly disappointed in the low R2 scores and believe that it is possible to further improve on our models.

First of all, an analysis of the data and abnormal cases, such as the tipis, can be done. This analysis could conceivably take many forms. One may, for example, choose to remove all listings that have 'nonsensical' values such as 'Tipi' or 'Yurt'. One may also choose to take these extreme cases and check them manually on Airbnb to find out what they actually are.

Secondly, the Airbnb listings contain more useful data that haven't been included in this research, such as review scores. Furthermore, each listing also has additional information like whether or not the unit has wifi or parking and so on. This information, while hard to parse, can be parsed and processed and used to create new relevant features.

Third, it is not necessary to consider models in isolation. One could consider applying ensemble methods, such as boosting or bagging, to the different models. Furthermore, one could consider mixing different models by generating a meta-model that uses a weighted average of different models to produce an output.

Finally, one could split the data differently. This research has opted for a 2/3 - 1/3 split between training and test data. It is also possible to split between training, validation, and test data. In the case, one could consider a 40 - 20 - 40 split, for example.

Bibliography

Acciani, Claudio, Vincenzo Fucilli, and Ruggiero Sardaro. "Data Mining in Real Estate Appraisal: A Model Tree and Multivariate Adaptive Spline Approach." *Aestimum* 58 (2011): 27-45. Web.

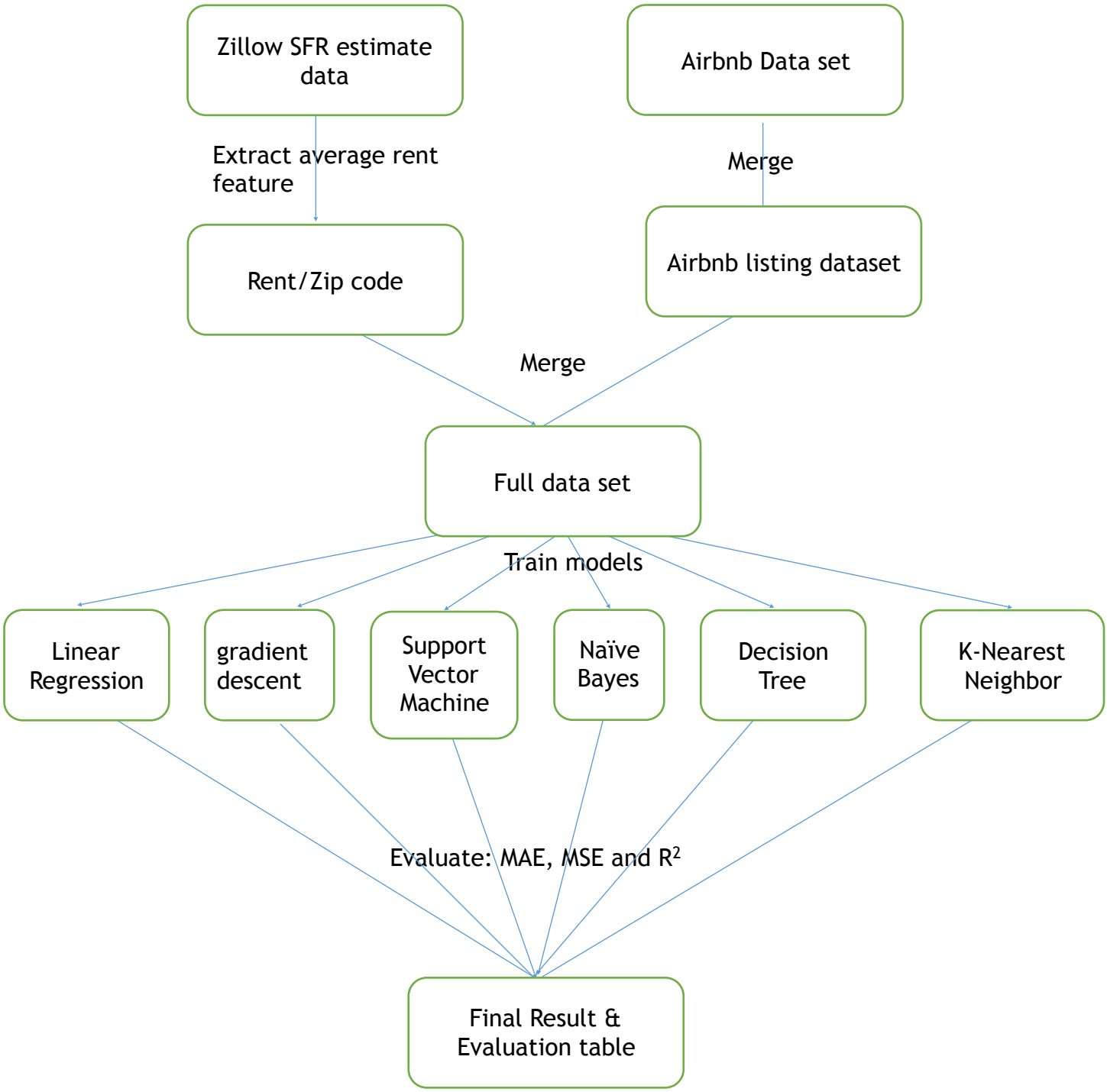
Kontrimas, Vilius, and Antanas Verikas. "The Mass Appraisal of the Real Estate by Computational Intelligence." *Applied Soft Computing* 11.1 (2011): 443-48. Web.

Stevens, Dick. "Predicting Real Estate Price Using Text Mining." Thesis. Tilburg University School of Humanities, Tilburg, The Netherlands, 2014. Web. <http://arno.uvt.nl/show.cgi?fid=134740>.

Tang, Emily, and Kunal Sangani. *Neighborhood and Price Prediction for San Francisco Airbnb Listings*. Web. <http://cs229.stanford.edu/proj2015/236_report.pdf>.

Zhao, Weikun, Cao Sun, and Ji Wang. "The Research on Price Prediction of Second-hand Houses Based on KNN and Stimulated Annealing Algorithm." *IJSH International Journal of Smart Home* 8.2 (2014): 191-200. Web.

Appendix A: Program Flowchart



Appendix B: Program Source Code With Documentation

---

train_models.py

```
#!/usr/bin/python
import sys
import csv

import matplotlib.pyplot as plt

import pydotplus

from sklearn import linear_model, tree
from sklearn.cross_validation import train_test_split
from sklearn.externals import joblib
from sklearn.externals.six import StringIO
from sklearn.feature_extraction import DictVectorizer
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import Normalizer
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor

# Build X and Y lists
# X : Features
# Y : Target
X = []
Y = []


with open('../Data/full_table.csv', 'r') as file:
    for line in csv.reader(file, delimiter = ','):
        if len(line) == 10:
            try:
                average_rent = float(line[2])
                property_type = line[3]
                room_type = line[4]
                accommodates = int(line[5])
                bathrooms = float(line[6])
                beds = int(line[7])
                bed_type = line[8]
                price = float(line[9])

                x = {
                    'average_rent': average_rent,
                    'property_type': property_type,
                    'room_type': room_type,
                    'accommodates': accommodates,
                    'bathrooms': bathrooms,
                    'beds': beds,
                    'bed_type': bed_type
                }

                y = price

                X.append(x)
                Y.append(y)
```

```
            except:
                pass


# The DictVectorizer converts data from a dictionary to an array
vectorizer = DictVectorizer()

# Convert X to Array
X = vectorizer.fit_transform(X).toarray()

# Store Vectorizer
joblib.dump(vectorizer, '../Models/vectorizer.pkl')

# Split X and Y into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33)

# Normalizer that will normalize the data
normalizer = Normalizer().fit(X)

# Normalized Features:
X_norm = normalizer.transform(X)
#X_norm = preprocessing.normalize(X)

# Store Normalizer
joblib.dump(normalizer, '../Models/normalizer.pkl')

# Split X and Y into training and testing sets for normalized data
X_norm_train, X_norm_test, Y_norm_train, Y_norm_test = train_test_split(X_norm, Y,
test_size=0.33)


output = []

output.append('Model, Mean Squared Error, Mean Absolute Error, R2 Score')


# Linear Regression
model = linear_model.LinearRegression()
model.fit(X_train, Y_train)
Y_pred = model.predict(X_test)
mse = mean_squared_error(Y_test, Y_pred)
mae = mean_absolute_error(Y_test, Y_pred)
r2 = r2_score(Y_test, Y_pred)

# Add to result to output
output.append('Linear Regression,{0},{1},{2}'.format(mse, mae, r2))

# Store model
joblib.dump(model, '../Models/linear_regression.pkl')


# Gradient Descent
model = linear_model.SGDRegressor()
model.fit(X_norm_train, Y_norm_train)
Y_pred = model.predict(X_norm_test)
mse = mean_squared_error(Y_norm_test, Y_pred)
mae = mean_absolute_error(Y_norm_test, Y_pred)
r2 = r2_score(Y_norm_test, Y_pred)

# Add to result to output
output.append('Gradient Descent,{0},{1},{2}'.format(mse, mae, r2))

# Store model
joblib.dump(model, '../Models/gradient_descent.pkl')


# Support Vector Machine Regression
model = SVR()
model.fit(X_train, Y_train)
Y_pred = model.predict(X_test)
mse = mean_squared_error(Y_test, Y_pred)
mae = mean_absolute_error(Y_test, Y_pred)
```

```
r2 = r2_score(Y_test, Y_pred)

output.append('Support Vector Machines,{0},{1},{2}'.format(mse, mae, r2))

# Store model
joblib.dump(model, '../Models/svm.pkl')

# Naive Bayes Regression
model = GaussianNB()
model.fit(X_train, Y_train)
Y_pred = model.predict(X_test)
mse = mean_squared_error(Y_test, Y_pred)
mae = mean_absolute_error(Y_test, Y_pred)
r2 = r2_score(Y_test, Y_pred)

# Add to result to output
output.append('Naive Bayes,{0},{1},{2}'.format(mse, mae, r2))

# Store model
joblib.dump(model, '../Models/naive_bayes.pkl')


# Decision Tree
mse_array = []
mae_array = []
r2_array = []
model_array = []

n_array = [ n + 1 for n in range(30)]

for n in n_array:
    model = DecisionTreeRegressor(max_depth=n)
    model.fit(X_train, Y_train)
    model_array.append(model)
    Y_pred = model.predict(X_test)
    mse = mean_squared_error(Y_test, Y_pred)
    mse_array.append(mse)
    mae = mean_absolute_error(Y_test, Y_pred)
    mae_array.append(mae)
    r2 = r2_score(Y_test, Y_pred)
    r2_array.append(r2)

mse, index = min((mse, idx) for (idx, mse) in enumerate(mse_array))
mae = mae_array[index]
r2 = r2_array[index]

model = model_array[index]

# Store model
joblib.dump(model, '../Models/decision_tree.pkl')

# Store representation of decision tree
dot_data = StringIO()
tree.export_graphviz(model, out_file=dot_data)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('../Figures/decision_tree_visualized.png')

output.append('Decision Tree(max-depth = {0}),{1},{2},{3}'.format(index, mse, mae, r2))

# Plot Evaluation metrics vs max-depth of decision tree
plt.figure()
plt.plot(n_array, mse_array, 'b-', label='Decision Tree')
plt.xlabel('Max Depth')
plt.ylabel('Mean Squared Error')
plt.legend()
plt.savefig('../Figures/decision_tree_mse.png')


plt.figure()
plt.plot(n_array, mae_array, 'b-', label='Decision Tree')
plt.xlabel('Max Depth')
plt.ylabel('Mean Absolute Error')
```

```
plt.legend()
plt.savefig('../Figures/decision_tree_mae.png')


plt.figure()
plt.plot(n_array, r2_array, 'b-', label='Decision Tree')
plt.xlabel('Max Depth')
plt.ylabel('R2 Score')
plt.legend()
plt.savefig('../Figures/decision_tree_r2.png')


# KNN
mse_array = []
mae_array = []
r2_array = []
model_array = []

n_array = [ n + 1 for n in range(50)]

for n in n_array:
    model = KNeighborsRegressor(n_neighbors=n)
    model.fit(X_train, Y_train)
    model_array.append(model)
    Y_pred = model.predict(X_test)
    mse = mean_squared_error(Y_test, Y_pred)
    mse_array.append(mse)
    mae = mean_absolute_error(Y_test, Y_pred)
    mae_array.append(mae)
    r2 = r2_score(Y_test, Y_pred)
    r2_array.append(r2)

mse, index = min((mse, idx) for (idx, mse) in enumerate(mse_array))
mae = mae_array[index]
r2 = r2_array[index]

model = model_array[index]

# Store model
joblib.dump(model, '../Models/knn.pkl')

output.append('KNN(#neighbors = {0}),{1},{2},{3}'.format(index, mse, mae, r2))

# Plot Evaluation metrics vs number of neighbors in KNN

plt.figure()
plt.plot(n_array, mse_array, 'b-', label='KNN')
plt.xlabel('Number of Neighbors')
plt.ylabel('Mean Squared Error')
plt.legend()
plt.savefig('../Figures/KNN_mse.png')


plt.figure()
plt.plot(n_array, mae_array, 'b-', label='KNN')
plt.xlabel('Number of Neighbors')
plt.ylabel('Mean Absolute Error')
plt.legend()
plt.savefig('../Figures/KNN_mae.png')


plt.figure()
plt.plot(n_array, r2_array, 'b-', label='KNN')
plt.xlabel('Number of Neighbors')
plt.ylabel('R2 Score')
plt.legend()
plt.savefig('../Figures/KNN_r2.png')

# Output the evaluation in a file called 'model_evaluation.csv'
with open('../Output/model_evaluation.csv', 'w') as file:
    csv_writer = csv.writer(file)
    csv_writer.writerows([row.split(',') for row in output])
```

demo.py

```
#!/usr/bin/python
import sys
import csv

from sklearn.externals import joblib

# Import the normalizer
normalizer = joblib.load('../Models/normalizer.pkl')

# Import the vectorizer
vectorizer = joblib.load('../Models/vectorizer.pkl')

# Import the models
linear_regression = joblib.load('../Models/linear_regression.pkl')
gradient_descent = joblib.load('../Models/gradient_descent.pkl')
svm = joblib.load('../Models/svm.pkl')
naive_bayes = joblib.load('../Models/naive_bayes.pkl')
decision_tree = joblib.load('../Models/decision_tree.pkl')
knn = joblib.load('../Models/knn.pkl')

# Setup the zip code vs rent table
zipcode_rent_table = {}
with open('../Data/full_zipcode_rent.csv') as file:
    for line in csv.reader(file, delimiter=','):
        if len(line) == 2:
            try:
                zipcode, rent = line
                rent = int(rent)
                zipcode_rent_table[zipcode] = rent
            except:
                pass



print('Zipcode, Property Type, Room Type, Accommodates, Bathrooms, Beds, Bed Type, Linear
Regression, Gradient Descent, SVM, Naive Bayes, Decision Tree, KNN')

# Read from the demo data
with open('../Data/demo_data.csv') as file:
    for line in csv.reader(file, delimiter=','):
        if len(line) == 7 and line[0] != 'Zipcode':
            try:
                zipcode, property_type, room_type, accommodates, bathrooms, beds, bed_type = line

                average_rent = zipcode_rent_table[zipcode]

                x = {
                    'average_rent': float(average_rent),
                    'property_type': property_type,
                    'room_type': room_type,
                    'accommodates': int(accommodates),
                    'bathrooms': float(bathrooms),
                    'beds': int(beds),
                    'bed_type': bed_type
                }

                # Consider the feature array
                x = vectorizer.transform(x).toarray()

                # Consider the normalized feature array (for gradient descent)
                x_norm = normalizer.transform(x)

                # Make predictions using each model
                linear_regression_pred = linear_regression.predict(x)[0]
                gradient_descent_pred = gradient_descent.predict(x_norm)[0]
                svm_pred = svm.predict(x)[0]
```

```
    naive_bayes_pred = naive_bayes.predict(x)[0]
    decision_tree_pred = decision_tree.predict(x)[0]
    knn_pred = knn.predict(x)[0]


    # Output the predictions
    output = ','.join([
        zipcode,
        property_type,
        room_type,
        accommodates,
        bathrooms,
        beds,
        bed_type,
        str(linear_regression_pred),
        str(gradient_descent_pred),
        str(svm_pred),
        str(naive_bayes_pred),
        str(decision_tree_pred),
        str(knn_pred)
    ])

    print(output)


except:
   pass
```

Appendix C: Input Output Listing

Full Table sample:

ID	Zipcode	Average Rent	Property Type	Room Type	Accommodates	Bathrooms	Beds	Bed Type	Price
10009751	10001	3908	Apartment	Entire home/apt	3	1	2	Real Bed	250
10035132	10001	3908	Apartment	Entire home/apt	4	1	2	Real Bed	195
10037605	10001	3908	Apartment	Entire home/apt	3	1	1	Real Bed	170
10058639	10001	3908	Apartment	Entire home/apt	8	1	2	Real Bed	225
10165577	10001	3908	Apartment	Entire home/apt	2	1	1	Real Bed	293
10191142	10001	3908	Cabin	Private room	1	3	1	Real Bed	60
10191182	10001	3908	Loft	Private room	3	3	3	Real Bed	200
10193426	10001	3908	Apartment	Entire home/apt	2	1	1	Futon	147
10210556	10001	3908	Apartment	Private room	2	2.5	3	Real Bed	175
10225827	10001	3908	Apartment	Entire home/apt	4	1	2	Real Bed	200
10271754	10001	3908	Apartment	Entire home/apt	4	1	2	Real Bed	100
10322621	10001	3908	Apartment	Private room	2	1	1	Real Bed	81
10377924	10001	3908	Cabin	Private room	1	3	1	Real Bed	62
10379070	10001	3908	Cabin	Private room	1	3	1	Real Bed	68
10379439	10001	3908	Cabin	Private room	1	3	1	Real Bed	62
10379735	10001	3908	Cabin	Private room	1	3	1	Real Bed	67
10380116	10001	3908	Cabin	Private room	1	3	1	Real Bed	63
10381513	10001	3908	Cabin	Private room	1	3	1	Real Bed	67
10381888	10001	3908	Cabin	Private room	1	3	1	Real Bed	67
10382377	10001	3908	Cabin	Private room	1	3	1	Real Bed	63

Zipcode vs Rent sample:

Zipcode	Rent
10025	3959
60657	3489
10023	4246
60614	5310
79936	1015
10002	4751

Evaluations with and without average rent feature:

Evaluation of Models with Average Rent Feature

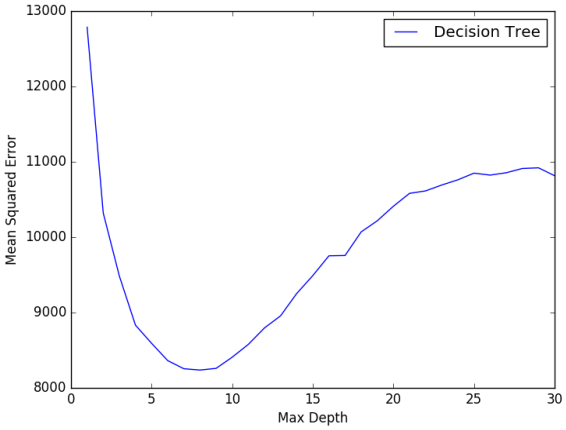
Model	Mean Squared Error	Mean Absolute Error	R2 Score
KNN(#neighbors = 9)	8013.354	51.478	0.528
Decision Tree(max-depth = 7)	8236.784	53.348	0.515
Linear Regression	8926.034	57.783	0.474
Support Vector Machines	14075.838	62.934	0.171
Gradient Descent	17016.612	87.120	0.000
Naive Bayes	39730.775	135.761	-1.341

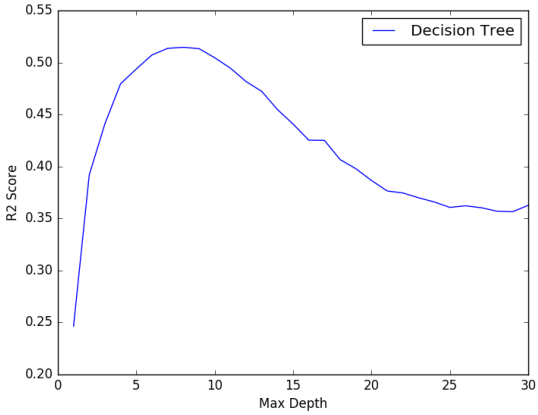
Evaluation of Models without Average Rent Feature

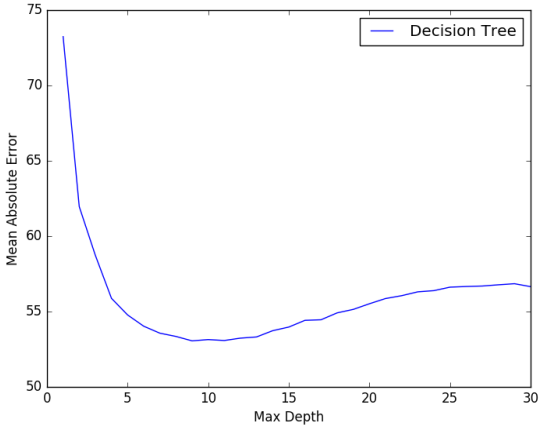
Model	Mean Squared Error	Mean Absolute Error	R2 Score
Decision Tree(max-depth = 5)	8710.870	56.698	0.472
KNN(#neighbors = 37)	8850.534	56.830	0.464
Support Vector Machines	9829.794	55.490	0.405
Gradient Descent	11692.555	68.772	0.300
Naive Bayes	129409.644	282.266	-6.838
Linear Regression	2205336894529920000000.	438046158.443	-133573133628885000.000

Appendix D: Model Graphs and Detail

Decision Tree Evaluation Metrics vs Max Depth:







KNN Evaluation Metrics vs Number of Neighbors: