HOUSE PRICE PREDICTION USING KERAS

Introduction

The trends in housing market not only affect the buyers and sellers, they reflect the current economic situation and social sentiments in the country. A house value is simply more than the location, square feet or number of bedrooms. There is a lot of things to be considered before fixing the house price. So, we are taking in all features available to predict the value of the house.

Outcome: This project will help in predicting house price around King County, USA, so we can have two types of clients:

1. House Buyers : They can use this model to identify whether the house price is reasonable or not. They can check which all features are contributing more to the house price and can have some compromise on it to get a affordable one.
2. House Sellers : With this model they can check the approximate price with which they can sell the house or they can invest more on features which will eventually increase the final price of the house.

The total process involved in house price prediction can be broken down to following steps:

1. Loading Data and Familiarisation
2. Data Analysing and Wrangling
3. Feature Scaling
4. Modelling and Prediction Using Keras

The software language used is Python and some major libraries like Numpy, Pandas, Keras (Modelling), Bokeh(Visualisation), Sklearn.

1. Loading Data And Familiarisation

Dataset consists of house prices from King County an area in the US State of Washington which is sold between May 2014 to May 2015. It consist of 21 variables (columns) and 21613 observations (rows).

Features available:

1. Id
2. Date (Sold date)
3. Price (Target variable)
4. Bedrooms
5. Bathrooms
6. Sqft_living

7. Sqft_lot
8. Floors
9. Waterfront
10. View
11. Condition
12. Grade
13. Sqft_above
14. Sqft_basement
15. Yr_built
16. Yr_renovated
17. Lat (Latitude)
18. Long (Longitude)
19. Sqft_living15
20. Sqft_lot15
21. Zip code

Loading Data

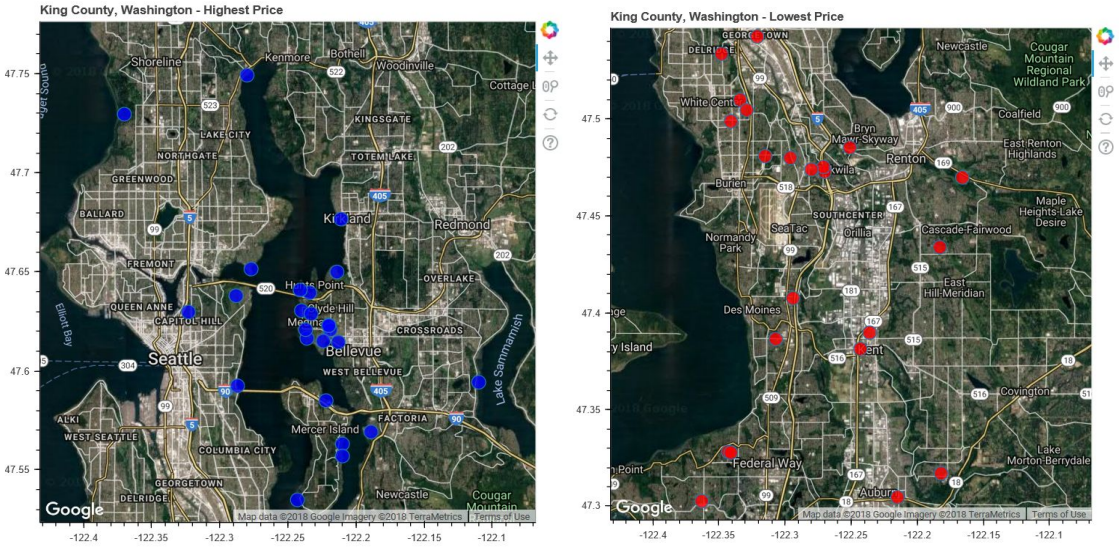
Data can be loaded using pandas **read_csv** function, giving PATH to the downloaded dataset.

```
import pandas as pd
pd.read_csv(PATH)
```

2. Data Analysing And Wrangling

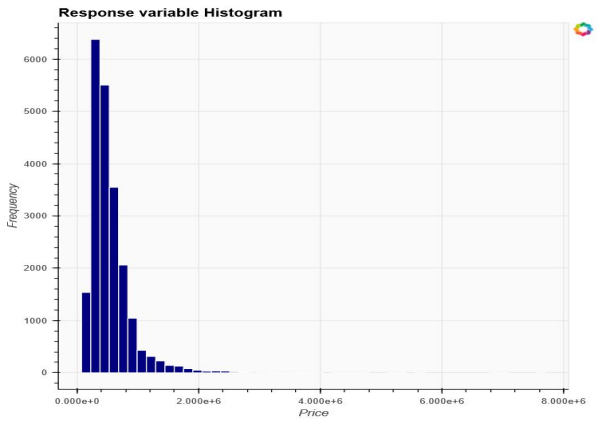
Analysis

1. Highest and lowest price by location in king county



The map shows the location of 25 houses having highest(blue) and lowest(red) prices in the king county, washington.

2. Histogram of the target variable (Price)



The histogram shows the price range of most of the house is around USD 20,00,000. Dataset has fewer houses having price greater than USD 40,000,00.

Wrangling

1. Checking for missing value

Missing values should be removed from the dataset or replaced with the mean value in the column.

2. Removing unwanted columns

As we know not all feature variable will be contributing to the response variable. So, we can avoid including those columns which have minor effect on response variable. Here, we can drop columns **date**, **zip code** and **id**.

3. Removing outliers

Outliers are data points which are very different from other data in the dataset. The major problem of having outliers are it can skew the result. It has significant effect on mean and standard deviation. So, it's better remove outliers from data.

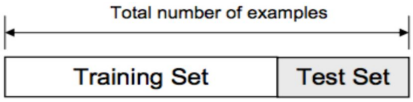
```
from scipy import stats
self.df = pd.DataFrame(self.df[(np.abs(stats.zscore(df_temp)) <= 4).all(axis=1)])
```

stats.zscore will calculate z score of each value in the sample relative to the sample mean and the standard deviation. Here we are removing rows having z score value greater than or equal to 4.

```
Number of Outliers: 706
Number of rows without outliers: 20907
```

4. Test train split

Splitting data set available to test and train, for the purpose of validation after modelling. We can split the dataset with 33% test data and remaining train data.




```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(training_set[features], prediction_set, test_size=0.33,
                                                    random_state=42)
# training_set[features] -  feature variables, prediction_set - Target variable price
```

3. Feature Scaling

Feature scaling is method used to standardize the range of feature or independent variable normally between 0 and 1. Here we have implemented scaling using min max scaler.
Min Max scaling is done using following equation:

$$X_{sc} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

```
from sklearn.preprocessing import MinMaxScaler
pre_processing_df = MinMaxScaler()
pre_processing_df.fit(mat_train) # mat_train - Train dataset
```

Above code snippet scales the train data, similarly we have to do for the test data.

4. Modelling and Prediction Using Keras

Deep learning is a sub field of machine learning and the algorithm tries to mimic human brain behaviour. Keras is one of the most popular deep learning library which is written in python. Keras can use different backends such as tensorflow, theano or CNTK. Here we are using tensorflow as our backend.
The entire process of modelling can be breakdown in to following steps:

1. Model creation
2. Compiling
3. Fitting
4. Evaluation

5. Prediction

1. **Model creation**

Terminologies:

(i) Will be using an Sequential model, it is a linear stack of layers.

(ii) Dense is fully connected layer that means all neurons in previous layers will be connected to all neurons in fully connected layer. Since we are predicting an output value the output layer will have a single neuron layer. i.e Dense(1)

(iii) BatchNormalization allows each to layer to learn a bit more independently. So, we can use it at output of each layer.

(iv) Dropout can be applied to both hidden and visible(input and output) layers. It's a simpler way of avoiding overfitting in the model. Dropout takes in percentage of neurons to be removed in the next layer.

(v) **Activation function:** Activation functions are used to introduce non linearity in data. There are different types of activation functions like Linear, Relu, Softmax, Tanh, Sigmoid etc. Here we are using Relu. It can be recognised as "needed" or "not needed". It zeros less needed signals and it can act as an excitatory neurons, it reacts on relevant signals and passes information to further layers.

The model has been chosen by trial and error method. The code snippet and the flow diagram of the model is shown below.

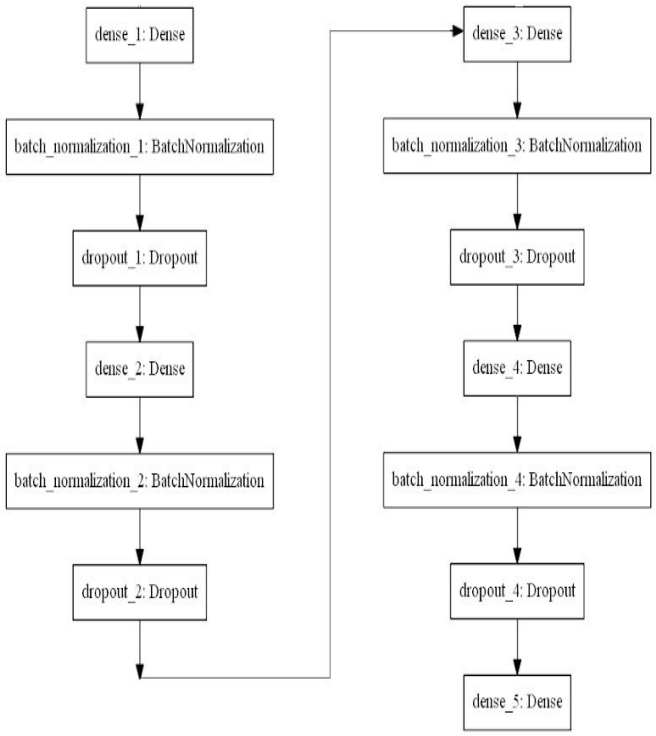
```
import keras
from keras.layers import Dense, Dropout, BatchNormalization

def create(self):
    self.model.add(Dense(32, activation='relu', input_shape=(17,)))
    self.model.add(BatchNormalization())
    self.model.add(Dropout(0.5))

    self.model.add(Dense(32, activation='relu', input_shape=(50,)))
    self.model.add(BatchNormalization())
    self.model.add(Dropout(0.5))

    self.model.add(Dense(32, activation='relu', input_shape=(50,)))
    self.model.add(BatchNormalization())
    self.model.add(Dropout(0.5))

    self.model.add(Dense(1))
```



2. **Compiling**

Compile models defines mainly **loss**, **optimizer** and **metrics**. Optimizer functions are used to modify the weights during back propagation(process of transmitting error backward to layers for modifying weights). Here we have used Adam as the optimizer and the loss function is mean square error. Metrics is used to judge the performance of the model. Model should be trained only after running the compilation.

```
def compile(self):
    self.model.compile(loss='mean_squared_error', optimizer=Adam(), metrics=['mse'])
```

3. **Fitting**

Fitting is done to train the keras model, parameters are the train data (X, Y), batch_size, epochs, verbose.

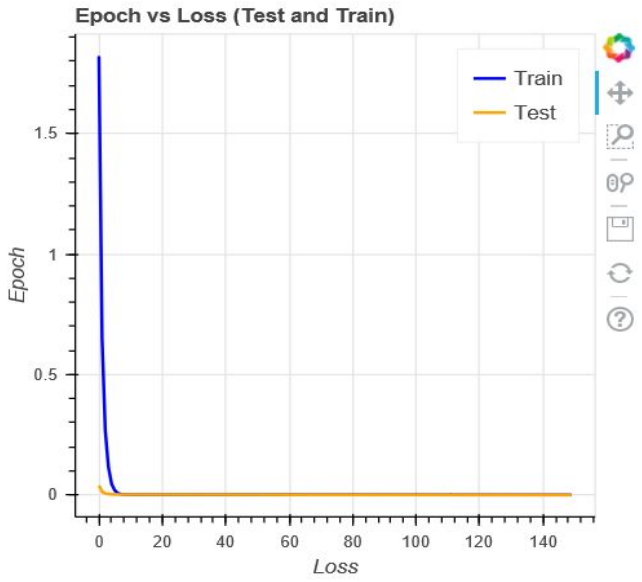
x_train: Train feature variables

y_train: Train output variables

epochs: number of iterations on a dataset

batch_size: batch of data taking in at a time

```
def fit(self):
    history = self.model.fit(self.x_train, self.y_train, batch_size=50, epochs=300, verbose=1,
                             validation_data=(self.x_test, self.y_test))
```



From the plot of loss, we can see that the model has comparable performance on both train and validation datasets. So, the model working is good.

4. **Evaluation**

We can check the total loss associated with the model using evaluate method in keras.

```
    def evaluate(self):
        score = self.model.evaluate(self.x_test, self.y_test, verbose=1)
        print("Test loss:", score[0])
```

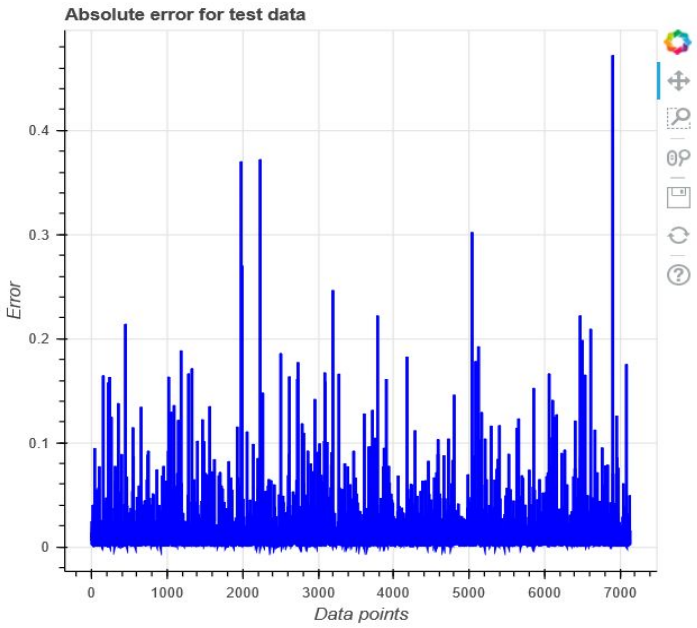
Test loss is:

```
Test loss: 0.0005422535813251496
```

5. **Prediction**

Predict method in keras will predict the output from the given test data.

```
def predict(self):
    y_pred = self.model.predict(self.x_test)
    return y_pred
```



Conclusion

The generated model was able to predict the house price with high accuracy. So, the model will equally work for both the house sellers and buyers. So that they will get an rough idea about the reasonable price to sell or buy the property.

Reference

The dataset was taken from kaggle,
https://www.kaggle.com/andyxie/regression-king-county-housing-price/data