Foundations of Data Science

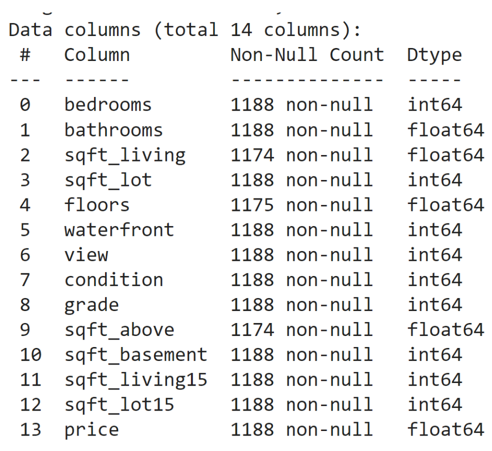
CS F320

Assignment 2

Team Members:	ID Numbers:
Vaibhav Mishra	2019A3PS1350H
Avinash Gondela	2019A8PS1357H
Akhilesh Senapati	2019AAPS1352H

1. Preprocessing the Data

The csv file contains 1188 rows and 14 columns with the following information:



```
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   bedrooms       1188 non-null   int64  
 1   bathrooms      1188 non-null   float64
 2   sqft_living    1174 non-null   float64
 3   sqft_lot       1188 non-null   int64  
 4   floors         1175 non-null   float64
 5   waterfront     1188 non-null   int64  
 6   view           1188 non-null   int64  
 7   condition      1188 non-null   int64  
 8   grade          1188 non-null   int64  
 9   sqft_above     1174 non-null   float64
 10  sqft_basement  1188 non-null   int64  
 11  sqft_living15  1188 non-null   int64  
 12  sqft_lot15     1188 non-null   int64  
 13  price          1188 non-null   float64
```

The price column is the target column whereas the other columns are the attributes.

Replacing NaN or Empty Values:

The above table clearly shows a few columns having less than 1188 entries which shows that our data has missing values. This can be due to data not collected properly or due to too little data. There are many ways to handle this data. Some include removing such rows or replacing those with some value.

In our model implementation, we replace NaN values with the average value of all the entries in that particular column thereby removing all NaN values.

We have taken the ceiling of all the values as some columns such as bedrooms, bathrooms can not take decimal values.

We have then standardized the data set by the following method:

```python
std_data=data
for i in data:
  std_data[i]=(data[i]-data[i].mean())/data[i].std()
std_data
```

We have taken each column from the data set and subtracted the mean of that column and divided by the standard deviation of all values of the data set.

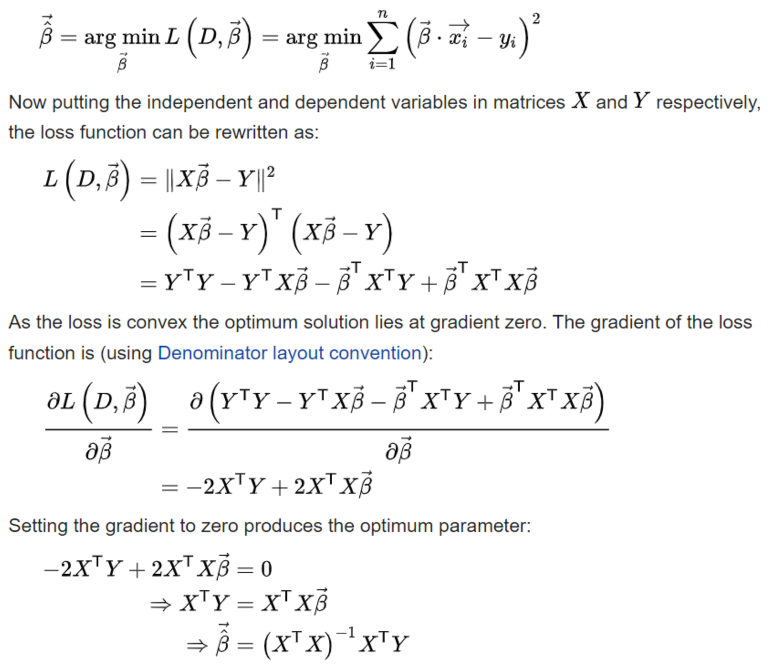
Removing Outliers:

After standardizing the data, we removed the outliers from the data set. Outliers are those observations which have abnormal distances as compared to the other points and also affect the training of the model. We removed the outliers i.e. we kept the points which lie within 2.5 range of standard deviation which give 99.37% of total data samples.

Dividing the Data set into Training and Testing:

The data has been divided into training and testing data. The training data contains 70% of the total data after removing outliers and the testing data contains 30% of the total data after removing the outliers. The target feature, price, has been removed from the testing data as well.

The below picture gives explanation of the Beta column matrix which we used for the implementation of the algorithm.



$$\vec{\hat{\beta}} = \underset{\vec{\beta}}{\arg\min}\, L\left(D, \vec{\beta}\right) = \underset{\vec{\beta}}{\arg\min} \sum_{i=1}^{n} \left(\vec{\beta} \cdot \overrightarrow{x_i} - y_i\right)^2$$

Now putting the independent and dependent variables in matrices X and Y respectively, the loss function can be rewritten as:

$$\begin{aligned} L\left(D, \vec{\beta}\right) &= \|X\vec{\beta} - Y\|^2 \\ &= \left(X\vec{\beta} - Y\right)^{\mathsf{T}} \left(X\vec{\beta} - Y\right) \\ &= Y^{\mathsf{T}}Y - Y^{\mathsf{T}}X\vec{\beta} - \vec{\beta}^{\mathsf{T}}X^{\mathsf{T}}Y + \vec{\beta}^{\mathsf{T}}X^{\mathsf{T}}X\vec{\beta} \end{aligned}$$

As the loss is convex the optimum solution lies at gradient zero. The gradient of the loss function is (using Denominator layout convention):

$$\begin{aligned} \frac{\partial L\left(D, \vec{\beta}\right)}{\partial \vec{\beta}} &= \frac{\partial \left(Y^{\mathsf{T}}Y - Y^{\mathsf{T}}X\vec{\beta} - \vec{\beta}^{\mathsf{T}}X^{\mathsf{T}}Y + \vec{\beta}^{\mathsf{T}}X^{\mathsf{T}}X\vec{\beta}\right)}{\partial \vec{\beta}} \\ &= -2X^{\mathsf{T}}Y + 2X^{\mathsf{T}}X\vec{\beta} \end{aligned}$$

Setting the gradient to zero produces the optimum parameter:

$$\begin{aligned} -2X^{\mathsf{T}}Y + 2X^{\mathsf{T}}X\vec{\beta} &= 0 \\ \Rightarrow X^{\mathsf{T}}Y &= X^{\mathsf{T}}X\vec{\beta} \\ \Rightarrow \vec{\hat{\beta}} &= \left(X^{\mathsf{T}}X\right)^{-1}X^{\mathsf{T}}Y \end{aligned}$$

2. Greedy Forward Feature Selection

Greedy Forward feature selection involves finding the optimal features from the given data frame which gives the optimal model for house price prediction.

Algorithm:
There are 12 features namely f1 to f12. The 13th feature is the price feature which is not included in the training as it is the target attribute. First we took each feature namely f1 to f12 at a time and built a model using y=wo+w1(f1) where wo and w1 are the coefficients. Similar models were built for all other features taken one at a time. We then found the Root Mean Square Error of each model whose formula is given by:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

The feature corresponding to the least RMSE value was picked and we repeat the above algorithm taking two features at a time. Similarly this was done 13 times and each time we stored the least RMSE for the next iteration. At the end of 13 iterations, we compared all the lowest RMSE values obtained and the minimum of those gives us the optimal number of features required for the price prediction.

Subset of Features that provide optimal model:

'waterfront', 'sqft_basement', 'floors', 'sqft_above', 'condition', 'bathrooms', 'sqft_lot', 'sqft_lot15', 'sqft_living15', 'bedrooms', 'grade', 'view', 'sqft_living

13 Features

Greedy forward code explanation:

A similar approach was taken with respect to greedy backward code explanation instead of deletion of features we add features one by one in each iteration.

3. Greedy Backward Feature Selection

Greedy Backward feature selection involves finding the optimal features from the given data frame which gives the optimal model for house price prediction.

Algorithm:

The algorithm is similar to that of greedy forward feature selection but the difference is backward meaning we start with all features. In this algorithm, we first eliminate one feature from f1 to f12 one at a time and find the RMSE. After calculating the RMSE, we store the least RMSE obtained and repeat the process but this time with deleting two features, one is from the previous iteration. This process is repeated for n iterations where n is the number of features. In our case n is 13. After all iterations, we find the minimum RMSE out of all the minimum RMSE obtained in each iteration and we drop the features accordingly.

Subset of Features that provide optimal model:

'waterfront', 'sqft_basement', 'floors', 'sqft_above', 'condition', 'bathrooms', 'sqft_lot', 'sqft_lot15', 'sqft_living15', 'bedrooms', 'grade', 'view', 'sqft_living

13 Features

Code Explanation:

In the inner loop, we first equated our feature matrix to the data set, which was inserted by a column of ones at index. This was due to the fact that in the next phase, we maintained eliminating a feature column while increasing the index with each iteration of the inner loop. Then we identified the beta column, which is the set of coefficients of the feature variable, in order to get the best optimised regression model, and then we calculated error using the aforementioned formula, and then we used an if condition to find the least rmse by deleting a feature at a time. The inner loop was completed, and the outer loop was completed by deleting the feature that yielded the lowest rmse and then moving on to the next iteration, recording root mean square values for both testing and training.

4. Table of Minimum Training and Testing Errors

The table below is as per the latest run

Model Name	Minimum Training Error	Minimum Testing Error
Greedy Forward Feature Selection	5.16515	443.8241
Greedy Backward Feature Selection	0.6282	25.89
Linear Regression	419566	4077418