QBUS6850

Machine Learning

For Business

2019 S1

GROUP ASSIGNMENT

GROUP 35

STUDENT ID:

470211966
470256521
470251663
480234847

Task A

(a)

The neural network is a mathematical model which mimics the structure and function of a biological neural network (especially the brain). It is used to estimate or approximate a function in most cases. In this question, the two main steps are divided in the process of implementation. One step is the forward propagation and another step is the backward propagation. The specific details would be displayed step-by-step as the following based on the figure1.

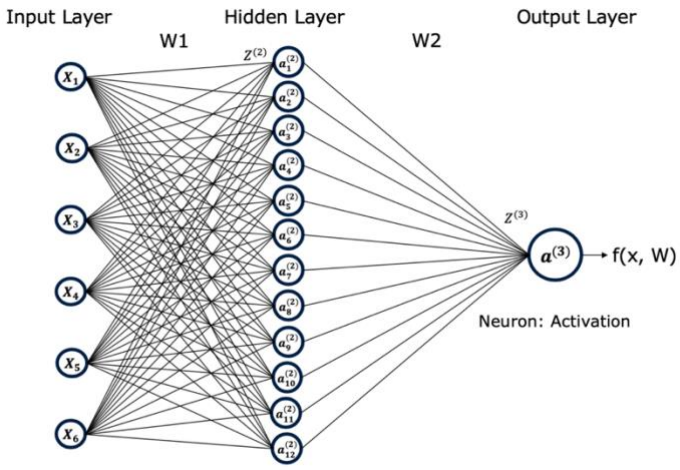

Figure 1 (3 layers neural network)

Step1: To begin with, feature scaling should be done with the help of python codes (see from figure2), dividing by maximum of x and y correspondingly. The aim is to increase the running speed and reduce the possibility of getting stuck in local optima, making sure features are on a similar scale.

```
import numpy as np
x = np.array(x)
y = np.array(y).reshape(-1,1)
x = x/np.max(x, axis = 0)
y = y/np.max(y, axis = 0)
```

Figure 2 (feature scaling by python codes)

Step2: We can see from the data that the input layer is a 100×6 array and $w^{(1)}$ is a 6×12 array. Here we have 72 weights, so $w^{(1)}$ has 72 elements. Firstly, $z^{(2)}$ could be calculated based on the formula below:

$$z^{(2)} = X \cdot w^{(1)}$$

Hence, $z^{(2)}$ is a 100×12 matrix and represents the total weighted sum of inputs to units in layer 2 now. Among the matrix, each row denotes one example and each column denotes one hidden unit. Then, we could apply activation function for each element of the $z^{(2)}$ matrix and generate $a^{(2)}$ based on the formulas below:

$$a^{(2)} = \sigma(z^{(2)}) = \frac{1}{1 + e^{-z^{(2)}}}$$

Step3: Similarly, we could generate $a^{(3)}$ according to the setp2 above. Since we have 3 weights, $w^{(2)}$ contains 3 elements. Thus, $w^{(2)}$ is a 12×1 array and $z^{(3)}$ could be produced in the light of the formula below:

$$z^{(3)} = a^{(2)} \cdot w^{(2)}$$

Therefore, $z^{(3)}$ is a 100×1 array now. Next, we can use the activation function to transform $z^{(3)}$ to $a^{(3)}$, that is, $f(x, W)$ in accordance with the formula as the following:

$$f(x, W) = a^{(3)} = \sigma(z^{(3)}) = \frac{1}{1 + e^{-z^{(3)}}}$$

In total, step2 and step3 indicate the implementation of forward propagation and $f(x,W)$ represents the final output here.

Step4: In order to minimize the loss function, from this step, we should use backward propagation to find the optimal $w^{(1)}$ and $w^{(2)}$ according to the use of chain rule. First of all, when we randomly set weights $w^{(1)}$ and $w^{(2)}$, we then execute forward propagation and calculate the loss. Next, the core part is that partial derivatives $\frac{\partial L(W)}{\partial w^{(2)}}$, $\frac{\partial L(W)}{\partial w^{(1)}}$ should be calculated. The specific calculations are as follows:

$$\frac{\partial L(W)}{\partial w^{(2)}} = \frac{\partial \frac{1}{2}(f(x,W)-t)^2}{\partial w^{(2)}} = \frac{\partial \frac{1}{2}(f(x,W)-t)^2}{\partial f(x,W)} \times \frac{\partial f(x,W)}{\partial z^{(3)}} \times \frac{\partial z^{(3)}}{\partial w^{(2)}}$$

$$= (f(x,W)-t) \cdot \frac{e^{-z^{(3)}}}{(1+e^{-z^{(3)}})^2} \cdot a^{(2)}$$

$$\frac{\partial L(W)}{\partial w^{(1)}} = \frac{\partial \frac{1}{2}(f(x,W)-t)^2}{\partial w^{(1)}} = \frac{\partial \frac{1}{2}(f(x,W)-t)^2}{\partial f(x,W)} \times \frac{\partial f(x,W)}{\partial z^{(3)}} \times \frac{\partial z^{(3)}}{\partial w^{(1)}}$$

$$= \frac{\partial \frac{1}{2}(f(x,W)-t)^2}{\partial f(x,W)} \times \frac{\partial f(x,W)}{\partial z^{(3)}} \times \frac{\partial z^{(3)}}{\partial a^{(2)}} \times \frac{\partial a^{(2)}}{\partial z^{(2)}} \times \frac{\partial z^{(2)}}{\partial w^{(1)}}$$

$$= (f(x,W)-t) \cdot \frac{e^{-z^{(3)}}}{(1+e^{-z^{(3)}})^2} \cdot w^{(2)} \cdot \frac{e^{-z^{(2)}}}{(1+e^{-z^{(2)}})^2} \cdot X$$

Step5: In this step, we could use the gradient descent below to update $w^{(1)}$ and $w^{(2)}$ with the help of python codes (see from figure3).

$$w^{(1)} = w^{(1)} - \alpha \frac{\partial L(W)}{\partial w^{(1)}}$$

$$w^{(2)} = w^{(2)} - \alpha \frac{\partial L(W)}{\partial w^{(2)}}$$

```
list1 = []
for i in range(0, 1):
    z2 = np.dot(x, W1)
    a2 = sigmoid(z2)
    z3 = np.dot(a2, W2)
    predict = sigmoid(z3)
    loss = 0.5 * sum((predict - y)**2)
    list1.append(loss)
    sigma3 = np.multiply((predict - y), sigmoid_derivative(z3))
    W2_derivative = np.dot(a2.T, sigma3)
    sigma2 = np.multiply(sigma3.dot(W2.T), sigmoid_derivative(z2))
    W1_derivative = np.dot(x.T, sigma2)
    W1 = W1 - alpha * W1_derivative
    W2 = W2 - alpha * W2_derivative
```

Figure 3 (gradient descent by python codes)

As a result, the optimal weights could be found in order to calculate the lowest loss function.

b)

The following figures illustrate the result of calculating for loss function value, $\frac{\partial L(W)}{\partial w^{(1)}}$, $\frac{\partial L(W)}{\partial w^{(2)}}$, $w^{(1)}$and $w^{(2)}$ for only 1 iteration under the python codes.

```
In [7]: loss.round(4)
Out[7]: array([15.2959])
```

Figure 4 (loss function value)

```
In [9]: W1_derivative.round(4)
Out[9]: array([[-0.6383,  1.7083, -0.5495,  0.7705,  0.5017,  0.3706,  0.354 ,
                -0.0517,  0.4256, -1.1752, -0.6106,  1.7554],
               [-0.4962,  1.3918, -0.449 ,  0.6292,  0.3801,  0.3011,  0.2878,
                -0.0433,  0.2895, -0.9652, -0.504 ,  1.4707],
               [-0.4989,  1.4313, -0.461 ,  0.6623,  0.4073,  0.3344,  0.2969,
                -0.0428,  0.3447, -0.9616, -0.5106,  1.4883],
               [-0.5451,  1.5023, -0.479 ,  0.6908,  0.4617,  0.3147,  0.2996,
                -0.043 ,  0.3641, -1.0227, -0.5276,  1.5301],
               [-0.7411,  1.9456, -0.6233,  0.8857,  0.5835,  0.4258,  0.402 ,
                -0.0585,  0.4906, -1.3341, -0.6927,  2.0162],
               [-0.4603,  1.1835, -0.3757,  0.5773,  0.3474,  0.2515,  0.2296,
                -0.0347,  0.2741, -0.812 , -0.4297,  1.2176]])
```

Figure 5 ($\frac{\partial L(W)}{\partial w^{(1)}}$value)

```
W2_derivative.round(4)
array([[-7.2726],
       [-3.8863],
       [-3.3535],
       [-5.8247],
       [-7.2394],
       [-1.0948],
       [-2.4187],
       [-6.5208],
       [-0.5856],
       [-3.1284],
       [-3.0764],
       [-5.8807]])
```

Figure 6 ($\frac{\partial L(W)}{\partial w^{(2)}}$value)

```
In [8]: W1.round(4)
Out[8]: array([[ 1.7641,  0.4   ,  0.9788,  2.2408,  1.8675, -0.9773,  0.9501,
                -0.1514, -0.1033,  0.4107,  0.1441,  1.4541],
               [ 0.7611,  0.1215,  0.4439,  0.3336,  1.494 , -0.2052,  0.313 ,
                -0.8541, -2.553 ,  0.6537,  0.8645, -0.7423],
               [ 2.2698, -1.4545,  0.0458, -0.1873,  1.5327,  1.4693,  0.1549,
                 0.3782, -0.8878, -1.9807, -0.3479,  0.1562],
               [ 1.2303,  1.2022, -0.3873, -0.3024, -1.0486, -1.42  , -1.7063,
                 1.9508, -0.5097, -0.438 , -1.2527,  0.7773],
               [-1.6138, -0.2129, -0.8954,  0.3868, -0.5109, -1.1807, -0.0282,
                 0.4283,  0.0665,  0.3026, -0.6343, -0.3629],
               [-0.6724, -0.3597, -0.8131, -1.7263,  0.1774, -0.4018, -1.6302,
                 0.4628, -0.9073,  0.052 ,  0.7291,  0.1289]])
```

Figure 7 ($w^{(1)}$ value)

```
W2.round(4)
array([[ 1.1401],
       [-1.2344],
       [ 0.4027],
       [-0.6842],
       [-0.8701],
       [-0.5787],
       [-0.3113],
       [ 0.0568],
       [-1.1651],
       [ 0.9011],
       [ 0.466 ],
       [-1.5357]])
```

Figure 8 ($w^{(2)}$ value)

c)

(c-1) Figure 9 shows the plot of loss function values with respect to each iteration. We can see from this graph that loss has been gradually going down with the iteration number increases. It is obvious that the line tends to be flat which approaches to x-axis when the iteration number reaches a certain amount.

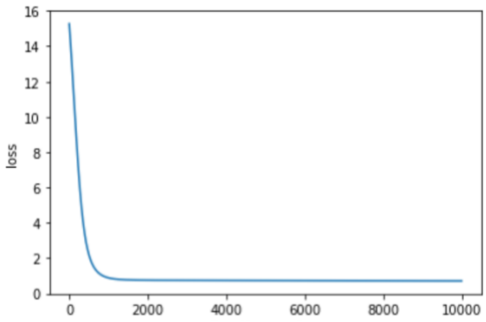

Figure 9

Figure 10 displays the final loss

```
In [13]: loss.round(4)
Out[13]: array([0.7082])
```

Figure 10

(c-2) Figure 11 and figure 12 illustrate the $w^{(1)}$value and $w^{(2)}$value

```
In [14]: W1.round(4)
Out[14]: array([[ 1.7783,  0.3338,  1.0084,  2.2073,  1.8442, -1.0022,  0.9367,
                 -0.1512, -0.1247,  0.4671,  0.1716,  1.3685],
                [ 0.7748,  0.059 ,  0.4734,  0.3057,  1.4757, -0.2268,  0.3008,
                 -0.8492, -2.569 ,  0.7088,  0.8928, -0.8216],
                [ 2.3255, -1.5803,  0.1144, -0.2352,  1.5001,  1.4328,  0.1345,
                  0.4015, -0.9271, -1.8627, -0.281 ,  0.0065],
                [ 1.2623,  1.1039, -0.3376, -0.3424, -1.0776, -1.4486, -1.7225,
                  1.962 , -0.5388, -0.3502, -1.2064,  0.6616],
                [-1.5681, -0.3348, -0.8332,  0.3361, -0.5466, -1.2188, -0.0495,
                  0.4438,  0.0271,  0.4143, -0.5743, -0.511 ],
                [-0.6629, -0.3993, -0.7962, -1.7494,  0.1623, -0.4177, -1.6382,
                  0.4614, -0.9219,  0.0849,  0.7462,  0.0765]])
```

Figure 11 ($w^{(1)}$ value)

```
W2.round(4)
array([[ 1.661 ],
       [-1.0537],
       [ 0.6137],
       [-0.3236],
       [-0.3933],
       [-0.4736],
       [-0.1557],
       [ 0.5362],
       [-1.1226],
       [ 1.0203],
       [ 0.6135],
       [-1.1802]])
```

Figure 12 ($w^{(2)}$ value)

(c-3) The figure below demonstrates the graph of the predict target values and observed target values. As we can see from the graph that the predicted target values almost fit well in the observed target values since the two broken lines nearly overlap.

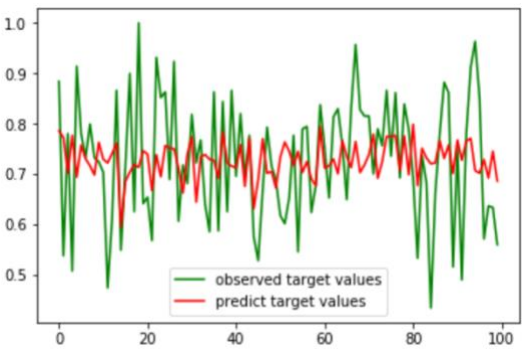

Figure 13

Task B

(a)

NMF is a new matrix decomposition algorithm that overcomes many problems of traditional matrix decomposition. The basic idea of NMF can be simply described as follow:

For any given non-negative matrix V, the NMF algorithm can find a non-negative matrix W and a non-negative matrix H. Thus, a non-negative matrix is decomposed into the product of these two non-negative matrices. Since the matrix before and after decomposition contains only non-negative elements, a column of vectors in the original matrix V can be interpreted as a weighted sum of all column vectors (called basis vectors) in the left matrix W, and the weight coefficient in the right called the matrix H.

In the process of decomposition in NMF, the formula is as the following:

$$\mathrm{V} \approx \mathrm{WH}$$

Then we could use the Euclidean distance square approach to do NMF loss function according to the formula below:

$$\mathrm{E} = \mathrm{V} - \mathrm{WH}$$

We expect to find the most optimal W and H to minimize $\| \mathrm{E} \|$. E complies with normal distribution, and we can get the maximum likelihood function as the follow:

$$\mathrm{L(W,H)} = \prod_{i,j} \frac{1}{\sqrt{2\pi}\sigma_{ij}} \exp\left(-\frac{E_{ij}^2}{2\sigma_{ij}}\right) = \prod_{i,j} \frac{1}{\sqrt{2\pi}\sigma_{ij}} \exp\left(-\frac{(V_{ij} - (WH)_{IJ})^2}{2\sigma_{ij}}\right)$$

After taking the logarithm, the log-likelihood function is given as follow:

$$\mathrm{lnL(W,H)} = \sum_{i,j} ln\frac{1}{\sqrt{2\pi}\sigma_{ij}} - \frac{1}{\sigma_{ij}} \times \frac{1}{2}\sum_{i,j}[V_{ij} - (WH)_{ij}]^2$$

Assume the variance of E of each data point remains the same, then the log-likelihood function should be maximized, and the value of the following objective function needs to be the lowest.

$$\mathrm{J(W,H)} = \frac{1}{2}\sum_{i,j}[V_{ij} - (WH)_{ij}]^2$$

Based on Euclidean distance:

$$(WH)_{ij} = \sum_k W_{ik}H_{kj} \qquad \frac{\partial (WH)_{ij}}{\partial W_{ik}} = H_{kj}$$

Hence:

$$\begin{aligned}\frac{\partial (WH)_{ij}}{\partial W_{ik}} &= \sum_j [\, H_{kj}(V_{ij} - (WH)_{ij})] \\ &= \sum_j V_{ij}H_{kj} - \sum_j (WH)_{ij}H_{kj} \\ &= (VH^T)_{ik} - (WHH^T)_{ik}\end{aligned}$$

Similarly, we can get:

$$\frac{\partial J(W,H)}{\partial H_{kj}} = (W^TV)_{kj} - (W^TWH)_{kj}$$

After that, we would use gradient descent to find the local minima:

$$W_{ik} = W_{ik} - \alpha_1 \times [(VH^T)_{ik} - (WHH^T)_{ik}]$$

$$H_{kj} = H_{kj} - \alpha_2 \times [(W^T V)_{kj} - (W^T W H)_{kj}]$$

Alpha is selected as follow:

$$\alpha_1 = \frac{W_{ik}}{(WHH^T)_{ik}} \qquad \alpha_2 = \frac{H_{kj}}{(W^T WH)_{kj}}$$

The final iteration formula is:

$$W_{ik} = W_{ik} \times \frac{(VH^T)_{ik}}{(WHH^T)_{ik}}$$

$$H_{kj} = H_{kj} \times \frac{(W^T V)_{kj}}{(W^T WH)_{kj}}$$

After completing the setting of the initial value of W_{ik} and H_{kj}, we could get the square loss of each iteration with the following formula:

$$\mathrm{L(W,H)} = \| V - WH \|^2 = \sum_{ij} (V_{ij} - (WH)_{ij})^2$$

(b)

The figure 14 shows the implement of multiplicative update algorithm of Nonnegative Matrix Factorization:

The Euclidean distance square loss with respect to each iteration is shown by figure 15:

```
LIST = []
for i in range(0, numIterations):
    H_kj1 = np.dot(W_init.T, V)
    H_kj2 = np.dot(np.dot(W_init.T, W_init), H_init)
    H_kj0 = H_kj1/H_kj2
    H = np.multiply(H_init, H_kj0)
    W_ij1 = np.dot(V, H.T)
    W_ij2 = np.dot(np.dot(W_init, H), H.T)
    W_ij0 = W_ij1/W_ij2
    W = np.multiply(W_init, W_ij0)
    W_init = W
    H_init = H
    distance = V - np.dot(W, H)
    distance_loss = np.sum(np.square(distance))
    LIST.append(distance_loss)
```

Figure 14

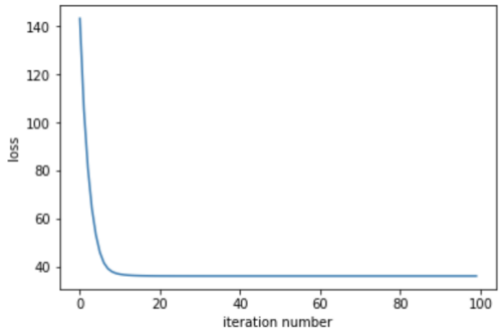

Figure 15

Obviously, our algorithm is converged based on the figure. Since the line gradually decreases with the iteration number increases and tend to be flat which approaches to the x-axis in the last.

(c)

The estimated W and H matrices with 100 times of iteration are given as figure 16:

```
print(W_init)

[[0.10091189 1.75014299]
 [0.64110297 1.96921335]
 [1.87023996 0.67155586]
 [2.65360314 0.09213938]
 [0.94828629 2.48059994]
 [0.21816624 2.47182374]]

print(H_init)

[[0.4257071  1.88040235 0.14243195 2.32482988 2.7585739 ]
 [2.48980961 0.79971177 1.87147071 0.07662044 1.49199463]]
```

Figure 16

Task C

1. Introduction

This report aims to develop several machine learning models to predict the sales price of the house in a certain area. We are provided with specific housing features with both numerical and categorical data including like general living area, total quality, neighborhood, etc. Five models including Lasso, Elastic Net, Gradient Boosting, XGBoost, and LightGBM will be applied to forecast the price. The pre-processing and feature engineering are also demonstrated in this report.

As a result, the model selection is based on the result of Root Mean Square Error (RMSE). Due to XGBoost performs the best in the end, we choose XGBoost as our final model.

2. Exploratory data analysis (EDA):

After importing the data, we first drop the 'Train ID' and 'Test ID' and combined the training set and testing set together to do preprocessing together.

2.1 Overall description of raw data:

To understand the dataset, we print the info of the dataset and find there are 80 features in training set and 79 features in testing set. Plus, there are 1398 samples in the training set and 1382 samples in the testing set. In addition, the dataset includes three data type: 'int64', 'object' and 'float64'. To be more specific, there are 43 categorical columns and 37 numerical columns in training data. Since the model package can only process the numerical data, we have to transform the categorical data into numerical one. Furthermore, proper method will be applied to handle the missing value.

2.2 Price Distribution:

To understand the target value, we plot the sales price and draw a normal distribution line accordingly to make a comparison. We also draw the probability line and calculate the skewness and kurtosis to analyze in detail, shown in figure 17 and figure 18.

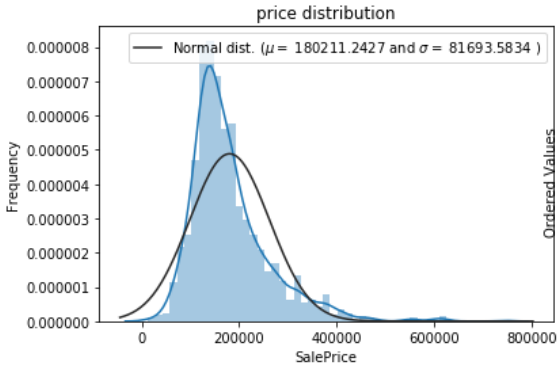

Figure 17

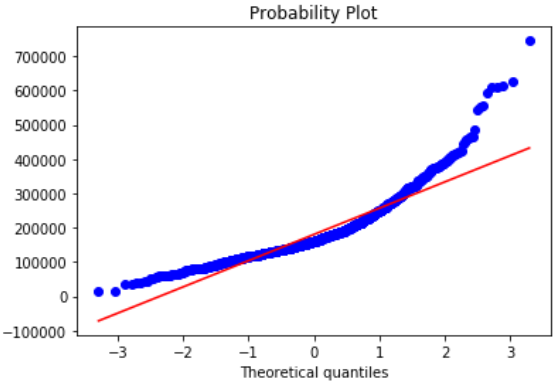

Figure 18

The skewness and kurtosis are calculated with python code, and the result is 1.7609 and 5.2898 respectively. We find that the mean sales price is 180211.2427 and the variance is 81693.5834. Since the figure is right-skewed, we decide to take a log transformation of the sales price. After the transformation, the figure is more approach to normal distribution. There are fewer outliers in probability line. The skewness and kurtosis of log-y is -0.1882 and 2.2095, which further proves that the target value is approximately normal distributed after transformation.

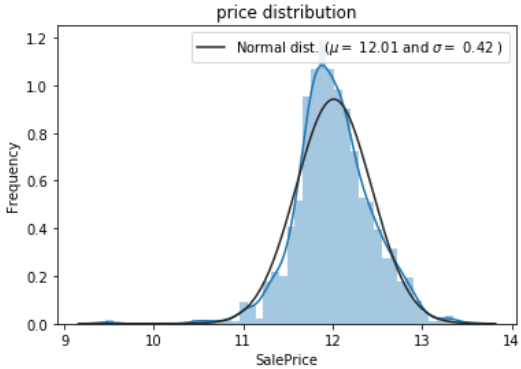

Figure 19

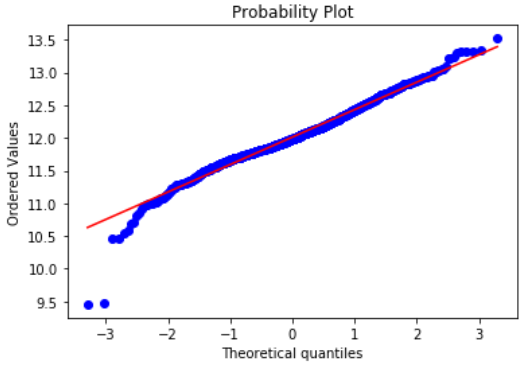

Figure 20

Numerical data

1)correlations between variables

Firstly, we plot the heatmap of the correlation coefficient between different variables and target value.

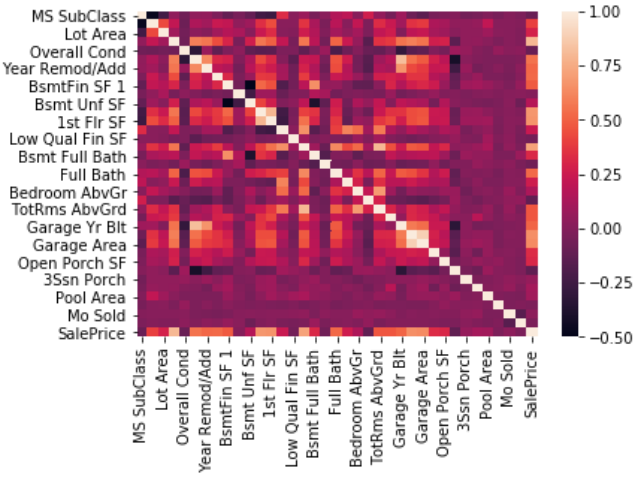

Figure 21

As a consequence, we found that '*Overall Qual*', '*Gr Liv Area*', '*Total Bsmt SF*' and '*1st Flr SF*' have the strongest correlation with the target value.

```
Top Absolute Correlations
Garage Cars     Garage Area      0.8939
Year Built      Garage Yr Blt    0.8238
Overall Qual    SalePrice        0.8030
Gr Liv Area     TotRms AbvGrd    0.7962
Total Bsmt SF   1st Flr SF       0.7876
dtype: float64
```

Figure 22

In terms of the correlation coefficient between variables, we found that 'Garage Cars' has a high correlation with 'Garage Area', this is because the size of the garage is usually designed considering its car capacity. The 'Year Built' is highly related to '*Garage Yr Blt*' which indicates that the garage is built when the building is constructed. What's more, '*Total Bsmt SF*' has high correlation with both '*1st Flr SF*' and '*2nd Flr SF*', this makes sense since the square area of a house in different floors is designed in the same scale. In most cases, the size of adjacent floors is highly related. Hence, in later analysis, we may create new features based on these correlated features.

2)correlations between log price and numerical variables

Since we have transformed the target value into log price, scatter plots are made to show the relationships between numerical variables and the log price.

For 'Gr Liv Area', we first plot the scatter against original price and find that most of the points follow a positive correlation with sales price except an outlier where its above ground living area is larger than 4000 while the sales price is lower than 300000, so we exclude this extreme value.

Then, we plot the scatter against log price find that the ground living area for most of the house is gathered around 1000-2000. The trend is approximately a positive linear regression.

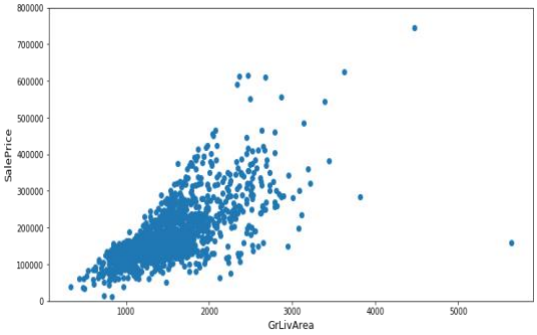

Figure 23

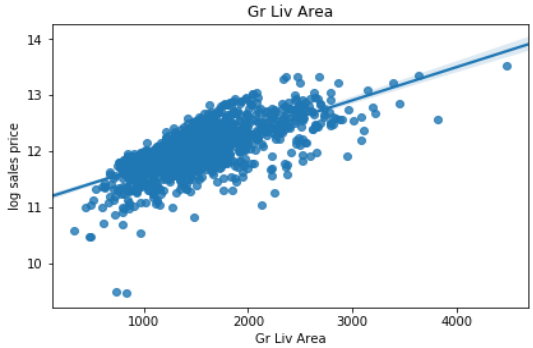

Figure 24

We take 'Overall Qual' as another example. It's highly related to the sales since its correlation

coefficient with the sales price is above 0.5. After log transformation, more data is fitted on the regression line. However, the number of outliers also increased in the front end since the scale of y decreased. The variance also decreased after log transformation.

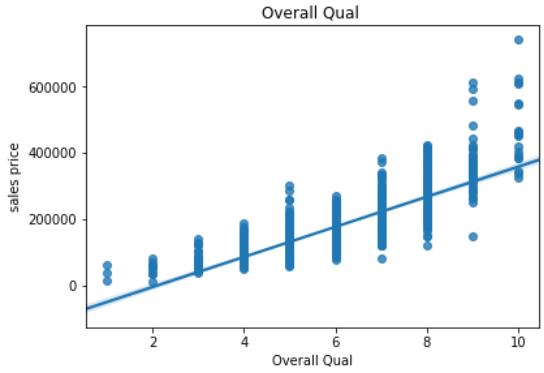

Figure 25

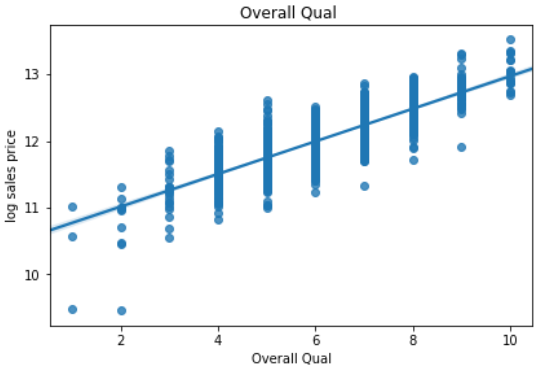

Figure 26

Categorical variables

In this case, we use boxplot to determine whether a categorical variable has significant effects on response. We can see that both these four categories have a great influence on the response. Take '*Paved Drive*' as an example, it's obvious that the average sale price of the house with the paved driveway is higher than partial pavement. House with the partial paved driveway is more expansive than one with gravel driveway. The sale price of house with paved driveway varied a lot and has a lot of outliers in a wide range which means only such house has the potential to be sold at a high price.

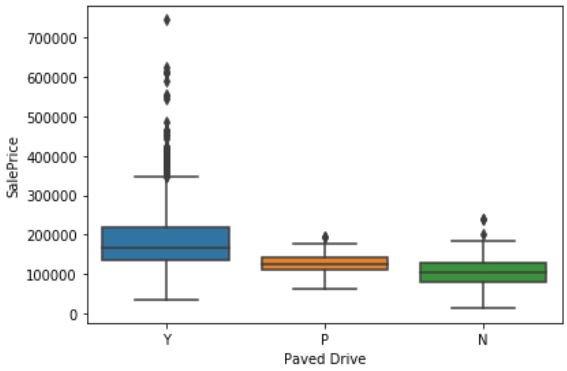

Figure 27

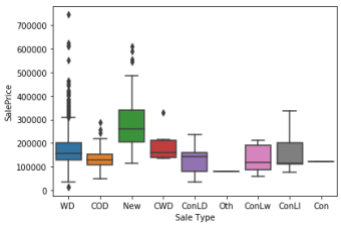

Figure 28

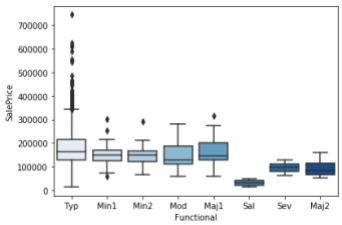

Figure 29

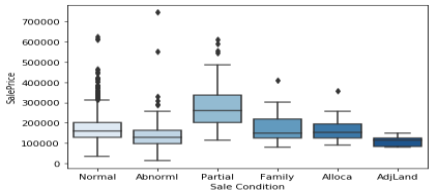

Figure 30

3. Data pre-processing

The data processing is about processing missing data, excluding outliers and creating new features. It can help avoid overfitting of the model, reduce the running time and improve the accuracy of prediction.

In data preparation, we randomly split 70% of the training dataset as the training set and the rest 30% as the validation set. Thus, we can evaluate the model by comparing the scoring metric result of the

validation set. We keep the consistent by process the training set and the validation set in the same way.

Missing value

NAN in traning set	**count**	**NAN in teat set**	**count**
Lot Frontage	253	Lot Frontage	217
Mas Vnr Type	14	Mas Vnr Type	8
Mas Vnr Area	14	Mas Vnr Area	8
Bsmt Exposure	2	Bsmt Qual	1
Electrical	1	Bsmt Cond	1
Bsmt Full Bath	1	Bsmt Exposure	2
Bsmt Half Bath	1	BsmtFin Type 1	1
Garage Yr Blt	89	BsmtFin SF 1	1
Garage Finish	1	BsmtFin Type 2	2
Garage Cars	1	BsmtFin SF 2	1
Garage Area	1	Bsmt Unf SF	1
Garage Qual	1	Total Bsmt SF	1
Garage Cond	1	Bsmt Full Bath	1
		Bsmt Half Bath	1
		Garage Yr Blt	65
		Garage Finish	1

The number of missing values has been listed in the table. We find that the columns that have missing values in the training set is not completely same as the test set. The common approaches of filling missing values are replacing NAN with a scaler value, filling gaps forward or backward and interpolation. In our case, we fill in median value for columns '*Lot Frontage*'. For columns "*Mas Vnr Area*", "*Bsmt Unf SF*", "*Total Bsmt SF*", "*Garage Cars*", "*BsmtFin SF 2*", "*BsmtFin SF 1*", "*Garage Area*" we use 0 to fill in NAN. This is because these missing data usually occurs when there is actually no such feature. For instance, the NAN in garage cars means there is no parking area for this house. For columns "*Pool QC*" , "*Misc Feature*", "*Alley*", "*Fence*", "*Fireplace Qu*", "G*arage Qual*", "*Garage Cond*", "*Garage Finish*", "*Garage Yr Blt*", "*Garage Type*", "*Bsmt Exposure*", "*Bsmt Cond*", "*Bsmt Qual*", "*BsmtFin Type 2*", "*BsmtFin Type 1*", "*Mas Vnr Type*" we use 'NA' to fill in missing values since these are all categorical variables. For columns "MS Zoning", *"Bsmt Full Bath", "Bsmt Half Bath", "Utilities", "Functional", "Electrical", "Kitchen Qual", "Sale Type","Exterior 1st", "Exterior 2nd"*, we use mode to fill in the NAN because these variables are usually in discrete numbers.

3.1 Feature Engineering

In feature engineering section, the original numerical features are reserved, and all categorical features are transformed into numerical data and dummy variables. For instance, the feature *MS SubClass* is transformed into one to six based on the *SalePrice* median. *Neighborhood, Condition 1, Bldg Type, Exterior 1st, Mas Vnr Type*, *MS Zoning*, and *Foundation* are in the same

transformation function with *MS SubClass* transformation. The basic idea to support us doing this transformation is that we perceive each item in categorical features has some size relationship with the sale price. Another type of feature contains *'Ex', 'Gd', 'TA', 'Po',* etc., which means the quality of that feature. We perceive that the better quality of that feature, the more contribution is on sale price. Specifically, *Fireplace Qu, Heating QC, Kitchen Qual, Exter Qual,* etc., these features are related with the quality of the entire house, so the quality mark has a positive relationship with the sale price. Two demonstrations of these two feature transformation are shown below. All categorical features are transformed under these two forms. For concise, this part only illustrates two features and detailed transformation is shown in python code.

Neighborhood		
'Landmrk':0		
'MeadowV':1		
'IDOTRR':2	'BrDale':2	
'OldTown':3		
'Edwards':4	'BrkSide':4	'Blueste':4
'Sawyer':5	'SWISU':5	
'NAmes':6	'NPkVill':6	
'Mitchel':7		
'SawyerW':8	'Gilbert':8	'NWAmes':8
'Blmngtn':8	'ClearCr':8	
'Greens':9	'CollgCr':9	
'Crawfor':10		
'Somerst':11		
'GrnHill':12	'Veenker':12	
'Timber':13		
'StoneBr':14		
'NoRidge':15		
'NridgHt':16		

However, we believe that each categorical feature has different contribution to sale price and the ordinal sort transformation based on median of sale price cannot reflect all implicit information, thus we transform all categorical variables into dummy variables.

Kitchen Qual
'Po':1
'Fa':2
'TA':3
'Gd':4
'Ex':5

In this part, we totally transform 80 features into 45 features, which is acceptable compared with the data size. Lasso regression is implemented to check the feature importance. The feature importance graph is shown below in the appendix. From the graph, *Overall Qual*, *Gr Liv Area* and *Year Built* are the most three important features to sale price, which is corresponding with real-life understanding.

With the better understanding of features, the combination of some features should have a more explicit relationship with sale price. For example, we believe the total house area including total basement area, first floor area, second floor area, and garage area should have a stronger correlation with sale price than those individual features mentioned above. Therefore, we create a new feature *TotalArea* to better represent those area features. Followed by this logic, some features like *Living Area Quality, Porch Area, Basement* are created using original features to present a stronger correlation with sale price with the hope of a good interpretation of features. After this step, PCA is implemented to decorrelate these newly created features since these features are highly correlated and lead to multicollinearity. In the end, total feature number is 430, which is in a reasonable feature range.

4.Methodology

In this section, we tend to use Lasso Regression, Elastic Net, Light GBM, XGBoost and Boosting models to predict the sale price. The advantages and disadvantages of these models will be explained in the following paragraph and based on that, we choose XGBoost and Boosting for our prediction. Then the definition and the function of these two models and the reason why we choose these two models in our prediction will be explicitly explained.

In insights and, specifically, in the fitting of linear or logistic regression models, the elastic net is a regularized method technique that directly consolidates the L1 and L2 punishments of the ridge and lasso strategies. Light GBM is an gradient boosting structure that utilize tree-based learning calculation. One of a kind characteristics of Light GBM is that it develops tree vertically, yet other tree-based calculation develops trees on a level plane. At the end of the day, Light GBM develops tree leaf-wise while other tree-based calculations develop level-wise. It will pick the leaf with max delta loss to develop. When developing a similar leaf, Leaf-wise calculation can lessen more loss than a dimension-wise calculation.

XGBoost, as a very state-of-the-art machine learning nowadays, is commonly used by data scientists to solve many data science problems quickly and accurately. Also, boosting Tree is one of the most broadly used algorithms in data mining and machine learning. Chen Tianqi(2016) demonstrates the scalability of XGBoost is contributed to several algorithm optimizations. These innovations are stated in the following parts. Firstly, it uses sparsity-aware split finding algorithm for sparse data and parallel tree learning. Many machine learning algorithms have no specific way to deal with sparse data, such as SVM, NN, etc. However, when XGBoost trains data where it uses data which is not missing to branch the node. Then we set out to put the missing data on the left and right nodes to see whether the missing value should be assigned to the left nodes or the right nodes. Secondly, a theoretically proper weighted quantile sketch allows managing instance weights in appropriate tree learning.

Furthermore, XGBoost improves a lot compared with GBDT. Traditional GBDT uses CART as the base classifier while XGBoost supports linear classifiers. In addition, GBDT uses only the first derivative information, but XGBoost performs a second-order Taylor expansion on the cost function and uses the first and second derivatives at the same time, pointing out the custom cost functions as long as they are derivable. One of the characteristics of which XGBoost beyond GBDT is that it adds a regularization term on the cost function to control the complexity of the model. Thus, the adjusted model is simpler and prevents overfitting. Moreover, it added shrinkage and column subsampling to prevent overfitting (Chen Tianqi,2016).

The objective function of XGBoost is as the following:

$$\mathcal{L}(\Phi) = \sum_{i=1}^{N} l(y_i, \hat{y}_i) + \Omega(f_k)$$

Amongst $l(y_i, \hat{y}_i)$ is error function and $\Omega(f_k)$ represents the regularization term which penalizes the complexity of each regression tree. Some targets could be used to measure the complexity of the tree such as the depth of the tree, the number of leaf nodes(T) and the scores of the nodes(W),

etc. XGBoost uses the regularization term formula below:

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \parallel w \parallel^2$$

The regularization term penalizes the leaf nodes and the scores of the nodes which is equivalent to pruning during the training process. After expanding the second-order Taylor expansion on the cost function, the loss function could be shown as the following:

$$\hat{L}^* = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

In terms of its system design, Chen Tianqi(2016) claims that XGBoost takes advantage of blocks for out-of-core computation. The first technique is the block compression which is compressed by columns and dynamically decompressed by independent threads when loaded into main memory. This helps to swap some of the calculations in the decompression with the disk read cost. Another technique is the block sharding which slices data onto multiple disks in another way, increasing the throughput of disk reads when multiple disks are available.

Gradient Boosting is one of the methods of Boosting. The main idea is that each time the model is built, it is established in the direction of the gradient of the loss function of the previously established model. The loss function describes the degree of unreliability of the model. The larger the loss function, the more error-prone the model is. If the model can make the loss function continue to decline, this means that it is constantly improving, and the best way is to let the function fall in the direction of its gradient.

Gradient Boosting can be described mathematically. Suppose the model can be represented by the following function. P represents a parameter that may have multiple parameters, like $P = \{P_0, P_1, P_2 \ ...\}$. F(x;P) is the prediction function, represents a function of x with P as a parameter. The model is a combination of multiple models. β represents the weight of each model, and α represents the parameters in the model.

$$\mathrm{F(x;P)} = F(x;\ \{\beta_m, \alpha_m\}_1^M) = \sum_{m=1}^{M} \beta_m h(x;\alpha_m)$$

$\phi(P)$ represents the likelihood function of P, at the same time, it is the loss function of the model F(x;P). We still use P to represent the parameters of the model, then we can get:

$$P^* = argmin(\phi(P))$$
$$\phi(P) = E_{Y,X}L(y, F(x;P))$$

Since the model (F(x;P)) is additive, for the parameter P, we can also get the following formula:

$$P^* = \sum_{m=0}^{M} P_m$$

In this way, the process of optimizing P can be a gradient descent process. Assuming m-1 models have been obtained so far if we want to get the m^{th} model, we need to evaluate the gradients of the first m-1 models and get the fastest direction of decline that represented by g_m.

$$g_m = \{g_{jm}\} = \{\left[\frac{\partial\phi(P)}{\partial P_J}\right]_{P=P_{m-1}}\}$$

And there is a very important assumption as follow. For the first m-1 models we find, we assume it is known, and it will not be changed. Our goal is to build the later model.

$$P_{m-1} = \sum_{i=0}^{m-1} P_i$$

The new model we get is in the gradient direction of the P-likelihood function. ρ is the distance that falls in the direction of the gradient.

$$P_m = -\rho_m g_m$$

Finally, we can get the optimal ρ by optimizing the following formula.

$$\rho_m = argmin\phi(P_{m-1} - \rho_m g_m)$$

Moreover, we can generalize the additivity of the parameter P to the function space, we can get the following function.

$$F_{m-1}(x) = \sum_{i=0}^{m-1} f_i(x)$$

$$g_m(x) = E_y[\frac{\partial L(y, F(x))}{F(x)}|x]_{F(x)=F_{m-1}(x)}$$

Hence, the expression of the m^{th} model:

$$f_m(x) = -\rho_m g_m(x)$$

In order to optimize F, {β, α}, which equals to P, has to be optimized. Calculate the loss function for the model F(x; α, β) for N sample points (xi, yi). The optimal {α, β} is the {α, β} that can make this loss function the smallest. $\{\beta_{m,}\alpha_m\}_1^M$ represents two m-dimensional parameters:

$$\{\beta_{m,}\alpha_m\}_1^M = argmin\sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \beta h(x_{i;}\alpha))$$

For each data point x_i, a $g_m(x_i)$ can be obtained, and finally we can get a complete gradient descent direction.

$$\overrightarrow{g_m} = \{-g_m(x_i)\}_1^N$$

$$-g_m(x_i) = -[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}]_{F(x)=F_{m-1}(x)}$$

In order to make $F_m(x)$ in the direction of $g_m(x)$, we can optimize the following equation by using the least squares method:

$$\alpha_m = argmin\sum_{i=1}^{N}(-g_m(x_i) - \beta h(x_{i;}\alpha))^2$$

Obtained on the basis of α, then can get β_m

$$\beta_m = argmin\sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \beta h(x_{i;}a_m))$$

Finally, we can get the model:

$$F_m(x) = F_{m-1}(x) + \rho_m\, h(x_{;}a_m)$$

The advantages and disadvantages can be summarized as follow:

Advantages:

• Usually provides predictive accuracy that cannot be exceeded.

• Great flexibility - Optimized for different loss functions and offers multiple hyperparameter adjustment options for maximum flexibility.
• No data preprocessing required - classifications and values are usually well used.
• Handling missing data - no interpolation required.
Disadvantages:
• GBM will continue to improve to minimize all errors. This may overemphasize the outliers and cause overfitting. Cross validation is required to neutralize.
• High flexibility causes many parameters to interact and seriously affect the behavior of the method (number of iterations, tree depth, regularization parameters, etc.). This requires a large grid search during the adjustment.
• Although this problem can be easily solved by various tools (variable importance, partial dependency graph, LIME, etc.), it is poorly interpreted.

In this way, to accomplish great outcomes in boosting calculation, the necessity for each tree is to be "weak" enough that the complication of every expansion is not enormous, and the all total number of trees is adequate. In XGBoost, the quantity of leaf nodes per tree is punished, which constrains the development of leaf nodes, making each tree "powerless", and furthermore presents learning rates, further lessening the effect of each tree. The expense of doing this is the all number of trees will be more, however from the point of view of its impact, it is advantageous. Also, the information is not huge for this forecast, so XGBoost performs well.

In modelling section, the train_test_split function is implemented to test the performance of the trained model. For XGBoost model, we choose four hyperparameters to tune, learning rate, estimator number, max depth and subsample. Then, randomized_search_CV is used to find the best parameters for XGBoost model. After that, we use split training data to fit the model and use split validation data to calculate the Rooted Mean Square Error. The same procedure with other four models is implemented, and the best two models are chose based on the RMSE score in the Kaggle competition.

5.Analysis

After feature engineering, training dataset has been applied to various models to do the model selection and get the predict price. The table below summarizes results of RMSE in Python and Kaggle. In order to search the best parameters, the randomized_search_CV tool has been used. In this case, Elastic Net model, XGBoost model, Light GBM model, Lasso model and Boosting model have been selected to generate predict price.

With comparison, in general, advanced models perform better than simple models. And among advanced models, the best-performed RMSE tested by our model in python is generated by XGBoost model and boosting model. Although there are slight differences between the RMSEs estimated by different dataset, this influences slightly to outcomes. Thus, our prediction seems relatively accurate. Consequently, the XGBoost model has been selected as our optimal model. In the

Model	RMSE in Kaggle
XGBoost	19088.86765
Boosting	19832.3839
LightGBM	20542.9745
Elastic Net	21723.4223
Lasso	20407.1103

following parts, performances of the best two models by using the same dataset will be discussed in detail.

5.1 Analysis of XGBoost

The XGBoost model has been chosen as the optimal model because it has the best RMSE score among all the models. The Figure below illustrates that *Bsmt Unf SF* and *Overall Condi* are the most important variables in XGBoost model.

A regular term is added to Xgboost to the cost function,and this regular term controls the model complexity in a decent level. The regular term contains leaf nodes number of each tree and the score output on each leaf node. In terms of Bias-variance tradeoff, the regular term reduces the variance of the model, which makes the learned model less complex and prevent overfitting This is also a advantage explaining xgboost best performance.

In comparison, when a negative loss is encountered during the split, the GBM will stop splitting;therefore, GBM is a greedy algorithm. XGBoost will split all the way to the specified maximum depth (max_depth) and then go back to pruning. If a node no longer has a positive value after it, it will remove this split. The advantage of this approach is that when a negative loss (such as -2) has a positive loss (such as +10), it appears. GBM will stop at -2 since it encountered a negative value. However, XGBoost will continue to split, and then find that the two splits will get +8 when combined, so these two splits will be preserved.

5.2 Analysis of Boosting

The Boosting model has been chosen as the optimal model because it has the second best RMSE score among all the models. The Figure below illustrates that *TotalHouse_OveralQual*, *GrLivArea_OverallQual* and *Neighborhood_TotalHouse* are the most important variables in XGBoost model. This variable importance figure proves that the idea of feature combination in feature engineering is effective and these features contribute much to sale price. This in some way explains the good performance of Boosting.

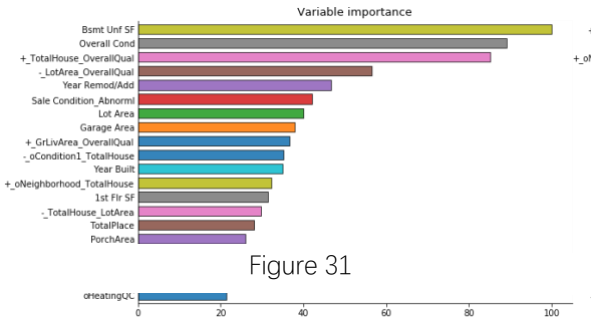

Figure 31

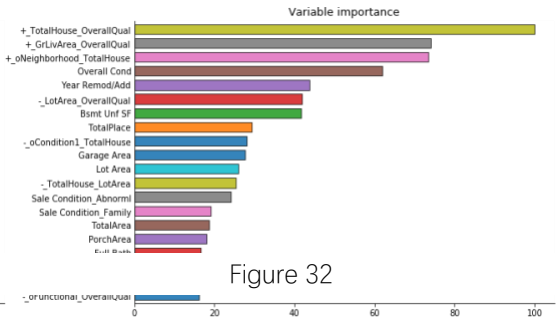

Figure 32

Gradient Boosting assembles an added substance model in a forward stage-wise design; it takes into consideration the advancement of arbitrary differentiable loss functions. In each stage, a regression tree is fit on the negative angle of the given loss work. The greatest advantage of Boosting is that the lingering computation of each progression really expands

the heaviness of the blunder occasion in mask, while the officially portioned example will lead in general to zero. This way the tree behind it can focus more and more on the instances of the previously misclassified.

5.3 Summary of analysis

Clearly, XGBoost outperformed Lasso and Elastic Net in the perspective of complexity and accuracy. Although Lasso and Elastic Net are relatively simple and have the advantages of readability and interpretability, XGBoost, Boosting and Light GBM, three advanced models could predict more accurate estimators than simple models. The advantages and disadvantages are explained in methodology section and the reason to choose XGBoost and Boosting is illustrated in this section.

6. Conclusion and limitations

In this report, we applied several graph plot methods to do exploratory data analysis. Then process the features according to their feature importance and correlations with each other by filling missing values, transforming categorical variables and creating new variables.

We finally applied 5 models to predict the price of the house and evaluate their performance by selecting the model with the least RMSE which stand for the accuracy of our prediction. We find that among the five models, XGBoost has the least MSE in the validation set. Therefore, we choose XGBoost as our final model and make predictions on the test set.

However, there are some limitations in our data pre-processing and feature engineering. For example, by filling NAN with mode, we may result in NAN if there is more than one mode in a certain variable. When doing feature engineering, we keep all the dummy variables of categorical data, which may result in overfitting and collinearity. Such errors may not be significant in the validation set but can cause a huge error in the test set. Furthermore, during the model adjustment process more parameters can be adjusted to make more precise prediction. Moreover, we can also try more methods to decrease the negative impact of outliers instead of doing a log transformation.

Appendix

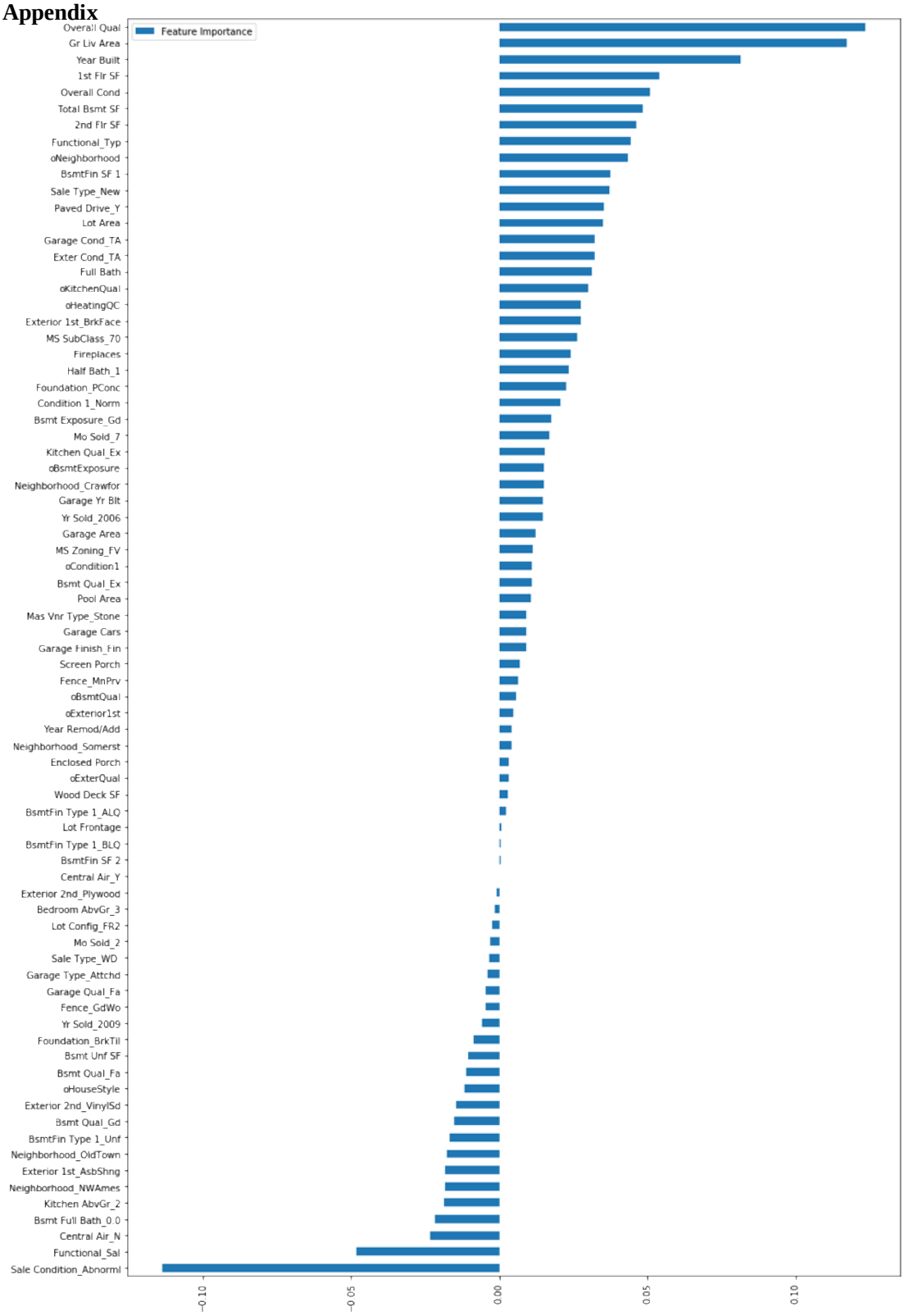

Figure1- Feature Importance run with Lasso

Reference

Chen Tianqi(2016). *XGBoost: A Scalable Tree Boosting System*

All You Need is PCA (LB: 0.11421, top 4%) | Kaggle. (2019). Retrieved from https://www.kaggle.com/massquantity/all-you-need-is-pca-lb-0-11421-top-4