1. Linear Regression

DataSet: kc_house_data

Source Code in “Linear_Regression.ipynb”, use Jupyter Notebook to open.

DataSet details

Take a quick look at the dataset and its data types.

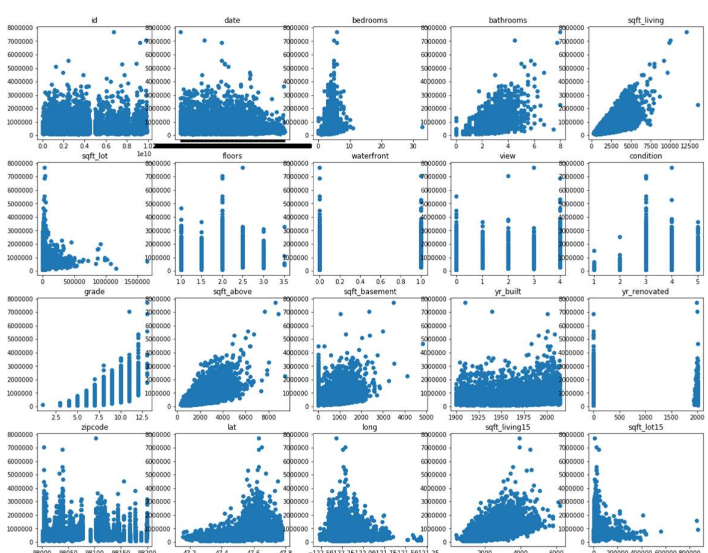
In [12]:
```
dataset.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
id               21613 non-null int64
date             21613 non-null object
price            21613 non-null float64
bedrooms         21613 non-null int64
bathrooms        21613 non-null float64
sqft_living      21613 non-null int64
sqft_lot         21613 non-null int64
floors           21613 non-null float64
waterfront       21613 non-null int64
view             21613 non-null int64
condition        21613 non-null int64
grade            21613 non-null int64
sqft_above       21613 non-null int64
sqft_basement    21613 non-null int64
yr_built         21613 non-null int64
yr_renovated     21613 non-null int64
zipcode          21613 non-null int64
lat              21613 non-null float64
long             21613 non-null float64
sqft_living15    21613 non-null int64
sqft_lot15       21613 non-null int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.4+ MB
```

n [13]:
```
dataset.head()
```
Out[13]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_b
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	1
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	1
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	1
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	1

5 rows × 21 columns

Visualize Analysis: Show the distribution of data.



Train/test split: Generally, use 90% of data for training, and 10% of data for testing. Shuffle the data first then split.

Algorithm Description

Data cleansing: I check the null values and 0 values in the dataset, the result shows there is no missing value in this dataset, so we don't need to do data cleansing.

Feature Choose: If we want to do linear regression, a linear relationship must be satisfied between the independent and dependent variables. From the pictures above, we can find some features: 'sqft_living','sqft_above','sqft_living15' Those features and 'price' can be seen to have a clear linear relationship. According to the meaning of these features, I chose 'sqft_living' as the feature to study.

Feature Scaling: We only select one feature to do linear regression, so there is no need to do scaling, but since the data is in the million orders of magnitude, if we don't do scaling, the program will overflow, so I still use the mean-std-normalization to scale the data. Also, I store the mean and std to help denormalization after prediction.

Algorithm Details:

Since we only have one feature, suppose there is a line $y = mx + b$ can fit data with minimal loss.

We use L2 loss. $Loss = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$

To minimize the error, we let the partial derivative of each variable is zero

$$\frac{\partial Loss}{\partial m} = \frac{\frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2}{\partial m} = -\frac{2}{N}\sum_{i=1}^{N} xi\big(y_i - (mx_i + b)\big) = 0$$

$$\frac{\partial Loss}{\partial b} = \frac{\frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2}{\partial b} = -\frac{2}{N}\sum_{i=1}^{N}\big(y_i - (mx_i + b)\big) = 0$$

We can define a very small "step" aka "learning_rate" to update m,b.

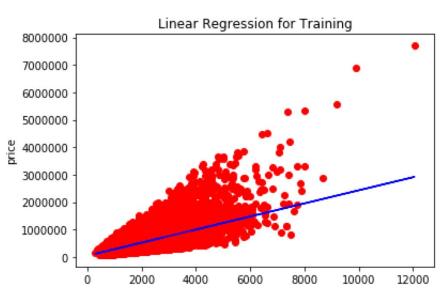
$$\mathrm{b} \;=\; \mathrm{b} + \mathrm{step} \times \frac{\partial Loss}{\partial b} \qquad \mathrm{m} \;=\; \mathrm{m} + \mathrm{step} \times \frac{\partial Loss}{\partial m}$$

verge, we can get the minimum loss.

Algorithm Results

Set learning_rate = 0.001, plot the line and scatter the real data. Calculate the root-mean-square.

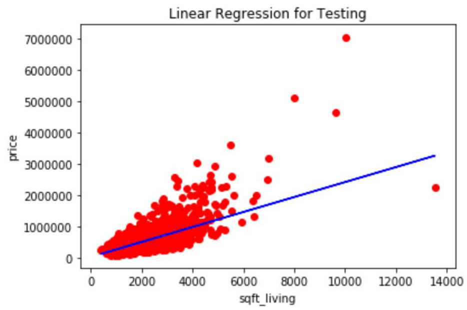
```
In [29]: plot(de_train,pred_train,'Linear Regression for Training','sqft_living','price')
         rmes(pred_train,de_train[:,1])

         root-mean-square is:
         261480.24000472383

Out[29]: 261480.24000472383
```



In [30]:
```
plot(de_test,pred_test,'Linear Regression for Testing','sqft_living','price')
rmes(pred_test,de_test[:,1])
```

```
root-mean-square is:
287889.6170322912
```

Out[30]:
```
287889.6170322912
```



The model performance can vary with learning rate.

```
for i in range(3):
    b,m = LinearRegression(train_set,learning_rate=0.001*pow(10,i))
    pred_test = pred(test_set[:,0],b,m,df_mean[1:],df_std[1:])
    de_test = denormalization(test_set,df_mean,df_std)
    print("learning_rate = ",0.001*pow(10,i))
    rmes(pred_test,de_test[:,1])
```

```
learning_rate =  0.001
root-mean-square is:
287889.6170322912
learning_rate =  0.01
root-mean-square is:
281683.1747995092
learning_rate =  0.1
root-mean-square is:
281682.99402821
```

Runtime

For each update, we need to traverse all data, calculate the gradient of every feature.
Each Iteration cost O(k*n) time. (K = feature size, n = data size)
Since we only have one feature here, so each iteration cost O(n) time
We don't know how many times the program will iterate until converge, it depends on learning rate and data itself.
Time complexity is O(iterationNumber * N).

The real wall time:

```
%%time
print("Training...")
b,m = LinearRegression(train_set)
```

```
Training...
Wall time: 36.9 s
```

2. Decision Tree

Consider the following set of training examples for the unknown target function $< X_1, X_2 > \rightarrow Y$.

Y	X_1	X_2	Count
+	T	T	3
+	T	F	4
+	F	T	4
+	F	F	1
-	T	T	0
-	T	F	1
-	F	T	3
-	F	F	5

1. What is the sample entropy $H(Y)$ for this training data (with logarithms base 2)?

$C_{(Y='+')} = 12 \qquad C_{(Y='-')} = 9$

$P_{(Y='+')} = \frac{12}{21} \quad P_{(Y='-')} = \frac{9}{21}$

$$\mathrm{H(Y)} = -\left(\frac{12}{21} \times \log_2 \frac{12}{21} + \frac{9}{21} \times \log_2 \frac{9}{21}\right) = 0.985$$

2. What are the information gains $IG(X_1) \equiv H(Y) - H(Y|X_1)$ and $IG(X_2) \equiv H(Y) - H(Y|X_2)$ for this sample of training data?

$$H(Y|X_1) = -\frac{8}{21} \times \left(\frac{7}{8} \times \log_2 \frac{7}{8} + \frac{1}{8} \times \log_2 \frac{1}{8}\right) - \frac{13}{21} \times \left(\frac{5}{13} \times \log_2 \frac{5}{13} + \frac{8}{13} \times \log_2 \frac{8}{13}\right) = 0.802$$

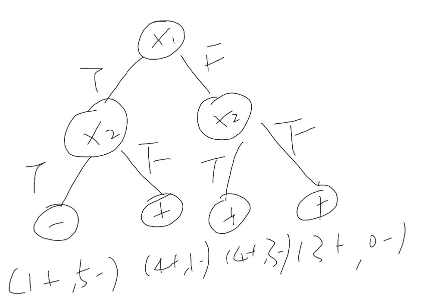
$$H(Y|X_2) = -\frac{10}{21} \times \left(\frac{7}{10} \times \log_2 \frac{7}{10} + \frac{3}{10} \times \log_2 \frac{3}{10}\right) - \frac{11}{21} \times \left(\frac{5}{11} \times \log_2 \frac{5}{11} + \frac{6}{11} \times \log_2 \frac{6}{11}\right) = 0.94$$

$IG(X_1) = 0.183$

$IG(X_2) = 0.045$

3. Draw the decision tree that would be learned by ID3 (without postpruning) from this sample of training data.

We want to maximize the IG so we choose X1 to be the first condition.



3. Perceptron

Use Jupyter Notebook to open file "Chuchu_Jin_python_HW2_Perceptron.ipynb"

4. Support Vector Machine

DataSet: wdbc.data

Source Code in “svm-smo.ipynb”, use Jupyter Notebook to oepn

DataSet details

Take a quick look at the dataset and its data types.

[3]:
```
df_wdbc.head()
```

Out[3]:

	ID	diagnosis	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	...	feature_20	feature_21	feature_22	feature_
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	201
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	195
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	170
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	56
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	157

5 rows × 32 columns

```
df_wdbc.shape
```

[4]: (569, 32)

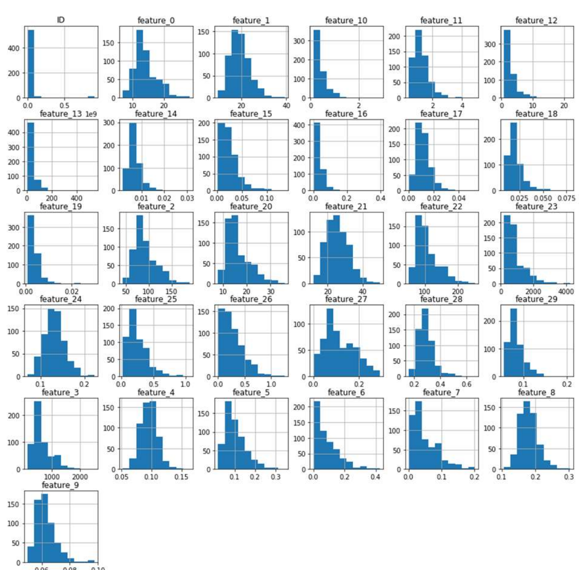
```
df_wdbc.describe()
```

[5]:

	ID	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	...	feature_20	feature_
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	...	16.269190	25.677
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	...	4.833242	6.146
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	...	7.930000	12.020
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	...	13.010000	21.080
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	...	14.970000	25.410
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	...	18.790000	29.720
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	...	36.040000	49.540

8 rows × 31 columns

Visualize Analysis: Show the distribution of data.



Train/test split: I choose 2/3 data for training, and 1/3 data for testing. Shuffle the data first then split.

Algorithm Description

Data cleansing: In file “wdbc.names” we know there is no miss value in this dataset, so we don’t need to do data cleansing.

Feature Scaling: As the data have very different range of value, we need to scale the data to make it easy to train.

Algorithm Details:

I use SMO to implement SVM.

$$\vec{w} = \sum_{i=1}^{N} y_i \alpha_i \vec{x}_i, \quad b = \vec{w} \cdot \vec{x}_k - y_k \text{ for some } \alpha_k > 0$$

ai means

$$\begin{aligned} \alpha_i = 0 &\Leftrightarrow y_i u_i \ge 1, \\ 0 < \alpha_i < C &\Leftrightarrow y_i u_i = 1, \\ \alpha_i = C &\Leftrightarrow y_i u_i \le 1. \end{aligned}$$

1. Choose the first a1 that obey the KKT condition, choose a random a2.
2. Calculate the upper and lower bound for a2
   - $L = \max(0, \alpha_2^{old} - \alpha_1^{old}), H = \min(C, C + \alpha_2^{old} - \alpha_1^{old}), if\ y_1 \neq y_2$
   - $L = \max(0, \alpha_2^{old} + \alpha_1^{old} - C), H = \min(C, \alpha_2^{old} + \alpha_1^{old}), if\ y_1 = y_2$
3. Update a2

$$E_i = u_i - y_i \quad \eta = K(\vec{x}_1, \vec{x}_1) + K(\vec{x}_2, \vec{x}_2) - 2K(\vec{x}_1, \vec{x}_2)$$

$$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\eta}$$

$$\alpha_2^{new} = \begin{cases} H, & \alpha_2^{new,unc} > H \\ \alpha_2^{new,unc}, & L \le \alpha_2^{new,unc} \le H \\ L, & \alpha_2^{new,unc} < L \end{cases}$$

4. Update a1

$$\alpha_1^{new} = \alpha_1^{old} + y_1 y_2(\alpha_2^{old} - \alpha_2^{new})$$

5. Update b

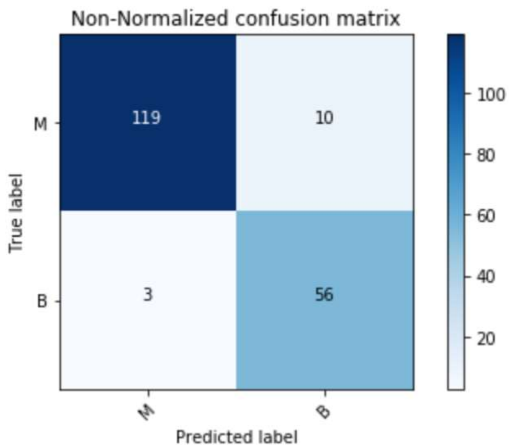
$$b_1^{new} = b^{old} - E_1 - y_1(\alpha_1^{new} - \alpha_1^{old})k(x_1, x_1) - y_2(\alpha_2^{new} - \alpha_2^{old})k(x_1, x_2)$$
$$b_2^{new} = b^{old} - E_2 - y_1(\alpha_1^{new} - \alpha_1^{old})k(x_1, x_2) - y_2(\alpha_2^{new} - \alpha_2^{old})k(x_2, x_2)$$

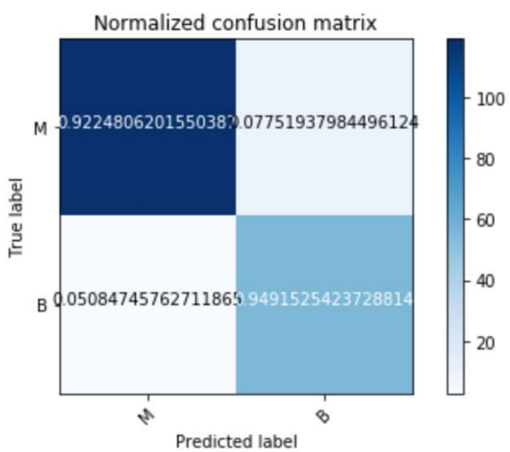
$$b = \begin{cases} b_1 & \text{if } 0 < \alpha_1^{new} < C \\ b_2 & \text{if } 0 < \alpha_2^{new} < C \\ (b_1 + b_2)/2 & \text{otherwise} \end{cases}$$

6. Loop Step 1-5 until alpha and b converge.

Algorithm Results

```
Accuracy is: 0.9308510638297872
Non-Normalized confusion matrix
[[119  10]
 [  3  56]]
Normalized confusion matrix
[[0.92 0.08]
 [0.05 0.95]]
```





Runtime

In each iteration,
For each alpha, (we have n alpha)
Firstly, we calculate w base on alpha, x and y, cost O(n) time
Then we check if the alpha(i) satisfy the KKT condition, if not, choose another alpha(j),update them.
So, each iteration cost O(n^2) time.
We don’t know how many times will it converge. Time complexity is O(iterationNumber * N^2).

The real wall time:

```
%%time
b,w = SMO(X_train.values,y_labels,1,10000,0.001)

CPU times: user 2h 26min 14s, sys: 5.54 s, total: 2h 26min 20s
Wall time: 2h 26min 15s
```