Implementing SGD on Linear Regression and comparing Manual SGD with sklearn SGD

In [0]:
```python
import warnings
warnings.filterwarnings('ignore')

#importing libraries and dataset
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.metrics import mean_squared_error
from prettytable import PrettyTable
from sklearn.linear_model import SGDRegressor
from sklearn import preprocessing
```

In [2]:
```python
from sklearn.datasets import load_boston
boston = load_boston()
print(boston.data.shape)
```

```
(506, 13)
```

In [3]:
```python
print(boston.feature_names)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

In [4]:
```python
print(boston.target.shape)
```

```
(506,)
```

In [5]:
```python
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usua
lly the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon Universit
y.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of
Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the
Tenth International Conference of Machine Learning, 236-243, University of Massachusetts, Amhers
t. Morgan Kaufmann.
```

In [0]:
```python
X = load_boston().data
Y = load_boston().target
```

In [7]:
```python
df=pd.DataFrame(X)
#some intuition
df[13]=df[10]//df[12]  #here we set a column 13 such that df[13]=Boston_data['Medv']//Boston_data
['B']
X=df.as_matrix()
df.head()
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	3.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	1.0
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	4.0
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	6.0
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	3.0

In [0]:
```python
#Splitting whole data into train and test
from sklearn.model_selection  import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X, Y, test_size=0.3, random_state=4)
```

In [9]:
```python
# applying column standardization on train and test data

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test=sc.transform(X_test)

df_train=pd.DataFrame(X_train)
df_train['price']=y_train
df_train.head()
```

Out[9]:

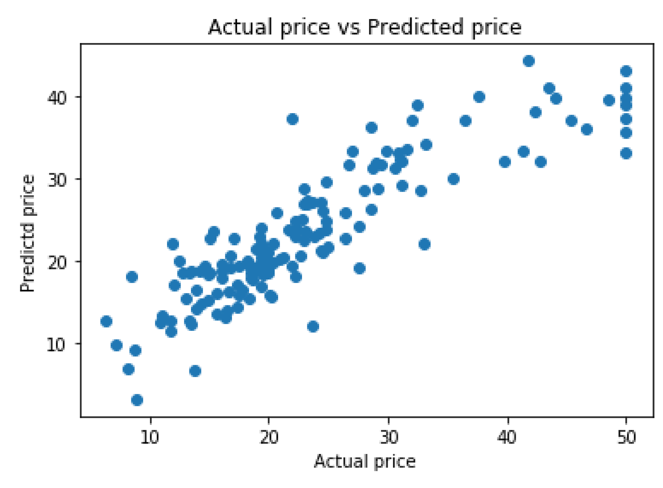
	0	1	2	3	4	5	6	7	8	9	
0	-0.425469	-0.470768	-0.954686	-0.231455	-0.919581	0.215100	-0.747410	0.454022	-0.764468	-0.976012	0.00
1	-0.426323	2.992576	-1.330157	-0.231455	-1.227311	-0.883652	-1.691588	3.163428	-0.651568	-0.464548	1.61
2	-0.385190	-0.470768	-0.705828	4.320494	-0.423795	-0.125423	0.818985	-0.353904	-0.199967	-0.623278	-0.5
3	-0.249268	-0.470768	-0.423497	-0.231455	-0.158805	-0.228336	1.021567	-0.021755	-0.651568	-0.623278	1.15
4	-0.365945	0.395068	-1.030363	-0.231455	0.157472	3.102729	-0.060078	-0.646202	-0.538668	-0.876071	-2.5

In [10]:
```python
#SGD Implementation for Lineat Regression.
#function having parameter X_train,y_train,no of iteration,learning rate r
#intialising no of iteration=100,learning rate =0.01
#batch size=20

W,B,iteration,lr_rate,k=np.zeros(shape=(1,14)),0,750,0.01,25 #intialise W and B to zero

while iteration>=0:
    w,b,temp_vectors,temp_intercept=W,B,np.zeros(shape=(1,14)),0
    data=df_train.sample(25) #sampling random k=batch size=20 data
    x=np.array(data.drop('price',axis=1))
    y=np.array(data['price'])

    for i in range(k):
        temp_vectors+=(-2)*x[i]*(y[i]-(np.dot(w,x[i])+b)) #partial differentiation wrt w dl/dw=1/k(-2
x)*(y-wTx-b)
        temp_intercept+=(-2)*(y[i]-(np.dot(w,x[i])+b)) #partial differentiation wrt b dl/db=1/k(-2)*
(y-wTx-b)

    W=(w-lr_rate*(temp_vectors)/k)
    B=(b-lr_rate*(temp_intercept)/k)

    iteration-=1


print(W)
print(B)
```

```
[[-1.24942365  0.89061473 -0.59881448  0.9203986  -1.40117399  1.55516223
   0.1680994  -2.91538048  2.19469597 -1.06691333 -2.2110781   0.92145648
  -1.96576631  2.71980831]]
[22.04635501]
```

In [0]:
```python
#prediction on x_test
#https://www.geeksforgeeks.org/numpy-asscalar-in-python/
y_predic_lr=[]
for i in range(len(X_test)):
    val=np.dot(W,X_test[i])+B  #val= wTx+b
    y_predic_lr.append(np.asscalar(val))
```

In [12]:
```python
#Scatter plot of actual price vs predicted price

plt.scatter(y_test,y_predic_lr)
plt.xlabel('Actual price')
plt.ylabel('Predictd price')
plt.title('Actual price vs Predicted price')
plt.show()
```

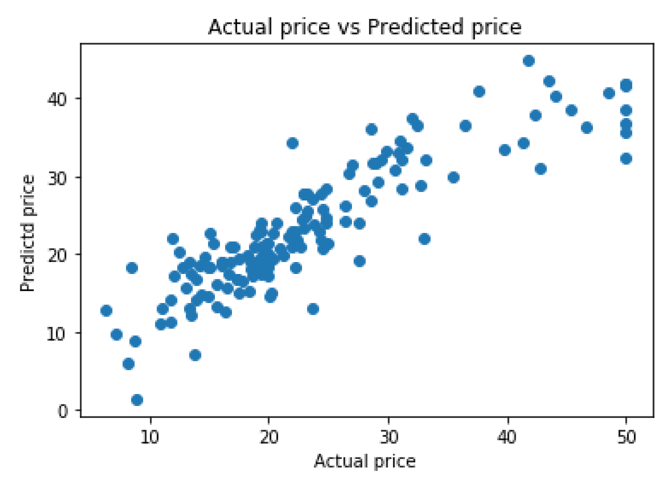


In [13]:
```python
MSE_lr=mean_squared_error(y_test,y_predic_lr)
print('mean squared error =',MSE_lr)
```

```
mean squared error = 23.914496793480303
```

In [17]:
```python
#SGD regression sklearn implementation

#intialising no of iteration=100,eta0=1
#taking t=2 and power_t=1 such that for each iteration eta0=eta0/pow(2,1) ,it means half each times

model=SGDRegressor(learning_rate='constant',eta0=0.01,penalty=None,n_iter=100,max_iter=100)
model.fit(X_train,y_train)
y_pred_sgd=model.predict(X_test)

#Scatter plot of actual price vs predicted price

plt.scatter(y_test,y_pred_sgd)
plt.xlabel('Actual price')
plt.ylabel('Predictd price')
plt.title('Actual price vs Predicted price')
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/stochastic_gradient.py:152: Deprecati
onWarning: n_iter parameter is deprecated in 0.19 and will be removed in 0.21. Use max_iter and t
ol instead.
  DeprecationWarning)
```



In [18]:
```python
MSE_sgd=mean_squared_error(y_test,y_pred_sgd)
print('mean squared error =',MSE_sgd)
```

```
mean squared error = 22.523397759886592
```

In [22]:
```python
#Comparison between weights obtained from manual implementation and weights obtained from sgd implem
entation
from prettytable import PrettyTable
x = PrettyTable()
x.field_names=['Weight vector manual','Weight vector SGD sklearn']
weight_sgd=model.coef_
for i in range(13):
    x.add_row([W[0][i],weight_sgd[i]])
print(x)
```

```
+----------------------+--------------------------+
| Weight vector manual | Weight vector SGD sklearn |
+----------------------+--------------------------+
| -1.2494236549195774  |     -1.447738141819672    |
|  0.8906147344020182  |     0.8937799376554265    |
| -0.5988144837915654  |    -0.40303205842820167   |
|  0.9203985964849772  |     0.3193745240501973    |
| -1.4011739908522907  |     -1.65965023682602     |
|  1.555162233635373   |     1.672760037743413     |
| 0.16809939759744807  |     0.4341658061223362    |
|  -2.915380476287036  |    -3.2413354813564643    |
|  2.1946959666080508  |     2.871108006032379     |
| -1.0669133272714628  |     -2.095579666719773    |
| -2.2110780960021876  |     -2.654728787069825    |
|  0.9214564800651012  |     1.0533008521707796    |
|  -1.965766307474707  |    -1.7118713082232575    |
+----------------------+--------------------------+
```

In [21]:
```python
#comparison between MSE of manual implementation and SGD sklearn implementation
print('MSE of manual implementation = ',MSE_lr)
print('-'*50)
print('MSE of SGD sklearn implementation = ',MSE_sgd)
```