In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
import warnings
warnings.filterwarnings("ignore")
from sklearn.datasets import load_boston
from random import seed
from random import randrange
from csv import reader
from math import sqrt
from sklearn import preprocessing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from prettytable import PrettyTable
from sklearn.linear_model import SGDRegressor
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from tqdm import tqdm
import os

from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
import pickle
from sklearn.datasets import load_boston
```

In [0]:

```
#loading the dataset
boston=load_boston()
X=boston.data
Y=boston.target

#splitting the dataset into train,test datasets
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=0)

#feature scaling
scaler=preprocessing.StandardScaler().fit(X_train)
X_train=scaler.transform(X_train)
X_test=scaler.transform(X_test)
```

```
X_test=scaler.transform(X_test)
```

In [3]:

```
print(X_train.shape,X_test.shape)
```

```
(354, 13) (152, 13)
```

SGDRegressor (SkLearn)

In [4]:

```
sklearn_model = SGDRegressor(penalty=None,loss='squared_loss',alpha=0,max_iter=1000,verbose=0)
sklearn_model.fit(X_train, Y_train)
```

Out[4]:

```
SGDRegressor(alpha=0, average=False, early_stopping=False, epsilon=0.1,
             eta0=0.01, fit_intercept=True, l1_ratio=0.15,
             learning_rate='invscaling', loss='squared_loss', max_iter=1000,
             n_iter_no_change=5, penalty=None, power_t=0.25, random_state=None,
             shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
             warm_start=False)
```

In [5]:

```
w_sklearn = sklearn_model.coef_
b_sklearn  = sklearn_model.intercept_
print("w",sklearn_model.coef_)
print("b",sklearn_model.intercept_)
```

```
w [-0.92067857  0.89159757 -0.19854314  0.64819434 -1.61035048  2.80469925
 -0.35168169 -2.93746844  1.32477611 -1.04757732 -2.2043927   0.58077839
 -3.36111457]
b [22.73861763]
```

MeanSquaredError for test-Data

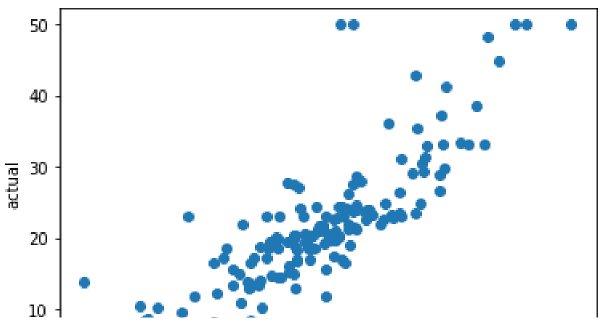
In [6]:

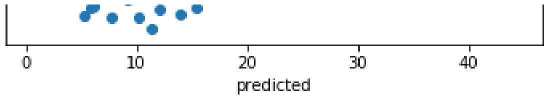
```
#mean squared error
Y_predicted=sklearn_model.predict(X_test)
sklearn_model_MSE=mean_squared_error(Y_test,Y_predicted)
print("Mean Squared Error ",sklearn_model_MSE)
```

```
Mean Squared Error  27.5665152774417
```

In [7]:

```
#predicted values vs actual values
plt.plot(Y_predicted,Y_test,linestyle='',marker='o')
plt.xlabel('predicted')
plt.ylabel('actual')
plt.show()
```





Custom Implementation of SGD

In [0]:

```
def loss(X,Y,W,b):
  Y_predicted=np.dot(X,W)+b
  mse_loss = mean_squared_error(Y,Y_predicted)
  return mse_loss
```

In [0]:

```
def SGDLinearRegression(X_train,Y_train,X_test,Y_test,W,b,r,iterations=2000,batch_size=128):
  n = len(Y_train)

  mse_train_list =[]
  MSE_test_list = []

  for j in range(iterations):
     #we are choosing random index's from X-train so that we can form batches from X_train
    index=np.random.choice(np.arange(len(X_train)),size=batch_size,replace=False)
    # X_batch has a set of random elements from X_train
    X_batch=X_train[index]

    Y_batch=Y_train[index]

    #  y and mean_squared_error is predicted at the start of each iteration which means for each epoch for Train DATA
    Y_predicted=np.dot(X_batch,W)+b
    mse_train = loss(X_batch,Y_batch,W,b)
    mse_train_list.append(mse_train)

    # We calculate the test mean_squared_loss for each updated Weights and Updated b_intercept
    Y_predicted=np.dot(X_test,W)+b
    MSE_test=mean_squared_error(Y_test,Y_predicted)
    MSE_test_list.append(MSE_test)

      #we could try if the difference between  mse_previous and mse_present value is less than 0.1 we need to stop the loop,
      #we could also try other metrics like if mse does not change for few iterations we could stop the outer loop


      # If the test loss is greater than SKLEARN_MODEL loss then Updated the Gradient,
      # If the test loss is less than SKLEARN_MODEL that is the Optimal MODEL. and Store the weights and Gradients
    if(MSE_test_list[j] > sklearn_model_MSE):

          # we need to calculated the loss of the first batch and then update the (w,b) parameters and then pass the second batch and so..on
          # We need to completer the first epoch and then calculate the MSE

          for i in range(batch_size):
            # calculating the y_predicted for first element in the batch and so..on
            y_predicted = np.dot(W.T,X_batch[i]) + b
            # updating the W parameter
            W = W - (-2)*r*X_batch[i]*(Y_batch[i] - y_predicted)
            # updating the b parameter
            b = b - (-2)*r*(Y_batch[i]-y_predicted)
          # we can decrease the learning rate based on weighted decay or exponential decay or based on the epoch number
          # but here we are decresing r for each iteration

          r = r/np.exp(j)
    else:
        break

  return W,b,mse_train_list,MSE_test_list
```

In [104]:

```
W=np.random.normal(0.0,1.0,size=13)
b=np.random.normal(0.0,1.0,size=1)
r = 0.01

print(W)
print(b)
W_updated,b_updated,mse_list,MSE_test_list = SGDLinearRegression(X_train,Y_train,X_test,Y_test,W,b,
r)
print(W_updated,b_updated)
```

```
[-0.20722756 -1.15267286 -0.01176299 -0.16369069  0.06363915  0.39520697
 -1.00796698  1.6082452   0.02039413 -0.44792601  0.58366876  1.11415582
  0.57167972]
[0.01485663]
[-0.1255768   0.82873766 -0.68481699  1.13840482 -0.93527466  2.97684526
  0.40426713 -2.19192893  1.27604899 -0.31087378 -1.72534143  0.6243121
 -4.45593436] [22.83036686]
```

In [105]:

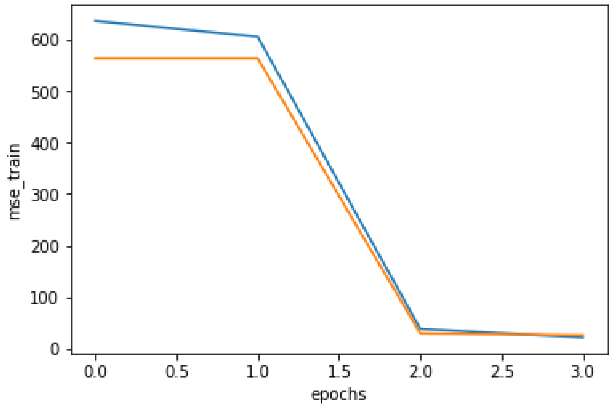
```
MSE_test_list
```

Out[105]:

```
[564.0665085299725, 564.0665085299725, 29.727041180223104, 26.371755430581246]
```

In [107]:

```
#predicted values vs actual values
n=4
plt.plot(range(0,n),mse_list[:n])
plt.plot(range(0,n),MSE_test_list[:n])
plt.xlabel('epochs')
plt.ylabel('mse_train')
plt.show()
```



In [108]:

```
Y_predicted=np.dot(X_test,W_updated)+b_updated
MSE_custom=mean_squared_error(Y_test,Y_predicted)
MSE_custom
```

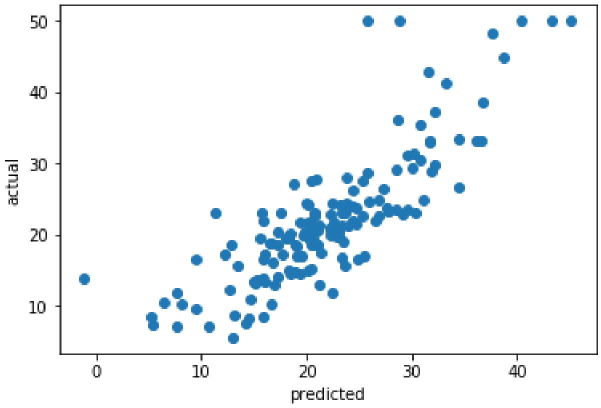
Out[108]:

```
26.371755430581246
```

In [109]:

```
#predicted values vs actual values
plt.plot(Y_predicted,Y_test,linestyle='',marker='o')
plt.xlabel('predicted')
```

```
plt.ylabel('actual')
plt.show()
```



In [110]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Weights", "sklear implementation", "my implementation"]


for i in range(13):
  x.add_row(["W"+str(i+1),w_sklearn[i],W_updated[i]])

x.add_row(["b",b_sklearn,b_updated])
print(x)
```

```
+---------+-----------------------+---------------------+
| Weights | sklear implementation |  my implementation  |
+---------+-----------------------+---------------------+
|    W1   |  -0.9206785659209031  | -0.1255767996530609 |
|    W2   |   0.891597567281791   |  0.8287376587046171 |
|    W3   |  -0.1985431353855247  | -0.6848169874669686 |
|    W4   |   0.6481943432228289  |  1.138404818728685  |
|    W5   |   -1.610350477195073  | -0.9352746581330771 |
|    W6   |   2.8046992476392436  |  2.976845256521955  |
|    W7   |  -0.3516816935278904  | 0.40426712504966494 |
|    W8   |  -2.9374684425134485  | -2.1919289305658056 |
|    W9   |   1.3247761126708053  |  1.276048994583615  |
|   W10   |  -1.0475773200174825  | -0.3108737825818627 |
|   W11   |  -2.2043927049301333  | -1.7253414316463378 |
|   W12   |   0.5807783859932566  |  0.6243120954743967 |
|   W13   |  -3.3611145726488365  |  -4.455934359649436 |
|    b    |     [22.73861763]     |    [22.83036686]    |
+---------+-----------------------+---------------------+
```

In [111]:

```
y = PrettyTable()

y.field_names = ["MSE", "Model"]
y.add_row([27.5665,"Sklearn Model"])
y.add_row([26.3717,"Custom Model"])
print(y)
```

```
+---------+---------------+
|   MSE   |     Model     |
+---------+---------------+
| 27.5665 | Sklearn Model |
| 26.3717 |  Custom Model |
+---------+---------------+
```