In [1]:

```python
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cross_validation import train_test_split
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
from sklearn.datasets import load_breast_cancer
import seaborn as sns
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
%matplotlib inline
```

```
C:\Users\SIRIL\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: Thi
s module was deprecated in version 0.18 in favor of the model_selection module into which all the
refactored classes and functions are moved. Also note that the interface of the new CV iterators a
re different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:

```python
cancer=load_breast_cancer()
```

In [3]:

```python
print(cancer.data.shape)
```

```
(569, 30)
```

In [4]:

```python
print(cancer.feature_names)
```

```
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

In [5]:

```python
cancer1=pd.DataFrame(data=cancer.data)
```

In [6]:

```python
cancer1.head(5)
```

Out[6]:

	0	1	2	3	4	5	6	7	8	9	...	20	21	22	23	24	
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	184.60	2019.0	0.1622	0.
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	158.80	1956.0	0.1238	0.
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	152.50	1709.0	0.1444	0.
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	98.87	567.7	0.2098	0.
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	152.20	1575.0	0.1374	0.

5 rows × 30 columns

Splitting the data

In [7]:

```
cancer_class=cancer.target
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(cancer1,cancer_class,test_size=0.33)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

Standardizing the data

In [8]:

```
scaler=StandardScaler()
X_train=scaler.fit_transform(np.array(X_train))
X_test=scaler.transform(np.array(X_test))
X_cv=scaler.transform(np.array(X_cv))
```

In [9]:

```
self_implement_train=pd.DataFrame(data=X_train)
self_implement_train['cancer_class']=y_train
```

In [10]:

```
from sklearn import linear_model
```

In [11]:

```
from sklearn import metrics
```

Using inbuilt SGD function

In [14]:

```
def inbuilt_sgd( n_iter=10000, X_train=X_train, X_test=X_test, Y_train=y_train, Y_test=y_test):
    sgd_clf=linear_model.SGDClassifier( loss='log', penalty='l2', alpha=0.0001,n_iter=n_iter)
    sgd_clf.fit(X_train, Y_train)
    y_pred=sgd_clf.predict(X_test)
    sgd_error=mean_squared_error(Y_test,y_pred)
    print('Number of iteration=', n_iter)
    a=metrics.accuracy_score(Y_test,y_pred)
    print("accuracy",a)
    return sgd_clf.coef_, sgd_clf.intercept_
```

case 1: with n_iter=10

In [15]:

```
w_sgd, b_sgd=inbuilt_sgd(n_iter=10)
```

```
Number of iteration= 10
accuracy 0.9414893617021277
```

weight vector for n_iter=10

In [16]:

```
print(w_sgd)
```

```
[[ -4.45401825  -4.47728884  -4.15115348  -4.51693729  -0.71912172
    9.71808087  14.63499163  26.99895586   6.77379789   6.68796825
```

```
   9.71808087 -14.83499183 -26.99895586  -6.77379789   6.88796825
 -37.26960164 -11.72923692 -28.35376648 -16.556478    11.43422718
  18.15227636  33.2780785  -18.81708289   0.93000574   9.3071154
 -11.61696868 -15.9332036  -10.27499479  -8.90223007  -1.01195161
  -9.7539044  -10.00466487 -21.26470189 -19.06833309 -11.03441007]]
```

bias/y_intercept for n_iter=100

In [17]:

```
print(b_sgd)
```

```
[-17.78849548]
```

case 2: with n_iter=100

In [18]:

```
w_sgd, b_sgd=inbuilt_sgd(n_iter=100)
```

```
Number of iteration= 100
accuracy 0.9521276595744681
```

weight vector for n_iter=100

In [19]:

```
print(w_sgd)
```

```
[[ 0.24997381  0.74730024  0.29788833 -0.30748721  0.85782373  5.04732264
  -4.47075971 -4.30562818 -0.94303578  2.29020391 -9.31665584 -1.46041007
  -6.55079199 -4.57535435  3.47373056  4.24410773  3.14178087 -3.57219819
   0.10260747  0.71790032 -2.07193363 -5.17229641 -1.76067353 -2.26519628
  -1.11297445 -0.61771744 -5.84695205 -2.67518487 -4.80697115  0.27218387]]
```

bias/y intercept

In [20]:

```
print(b_sgd)
```

```
[-3.75861114]
```

Self implementation of Logistic regression using SGD and l2 regularization

Defining Sigmoid function

In [21]:

```
import math

def sigmoid(x):
    if x<0:
        return 1 - (1 / (1 + math.exp(x)))# due to large negative x value
    else:
        return 1/(1+math.exp(-x))
```

defining w_gradient

In [22]:

```
def w_gradient(x,y,w_old,s,a,l,b_old):
    a=(1-(a*l/s))*w_old
    c=np.dot(x,w_old.T)+b_old
    b=a*x*(y-sigmoid(c))
    return a+b
```

defining b_gradient

In [23]:

```
def b_gradient(x,y,w_old,s,a,l,b_old):
    a=(1-(a*l/s))*b_old
    c=np.dot(x,w_old.T)+b_old
    b=a*(y-sigmoid(c))
    return a+b
```

the default value of 0.0001 we will take for alpha of regularization nd we use adaptive learning rate too

In [24]:

```
def self_implement_log_reg(X, lr_rate=1, n=1):
    w_new=np.random.normal(0,1,[1,30])
    b_new=np.random.normal(0,1,[1,1])
    k=1
    r=lr_rate
    L=0.0001
    a=0.001
    l=0.001
    while(k<=n):
        w_old=w_new
        b_old=b_new
        w_grad=np.random.normal(0,1,[1,30])
        b_grad=np.random.normal(0,1,[1,1])
        x_new=X.sample(15)
        x=np.array(x_new.drop('cancer_class',axis=1))
        y=np.array(x_new['cancer_class'])
        s=x_new.shape[0]
        for i in range(15):
            w_grad=w_grad+w_gradient(x[i],y[i],w_old,s,a,l,b_old)
            b_grad=b_grad+b_gradient(x[i],y[i],w_old,s,a,l,b_old)
        w_new=w_old-r*(w_grad+(2*L*w_old))#adding regularization term
        b_new=b_old-r*(b_grad)
        k+=1
        r=r/2
    return w_new,b_new
```

In [25]:

```
def prediction(x,w, b):
    y_pred=[]
    for i in range(len(x)):
        y=np.asscalar(np.dot(w,x[i])+b)
        y_pred.append(y)
    return np.array(y_pred)
```

Case1: n_iter=10

In [26]:

```
w,b=self_implement_log_reg(self_implement_train, lr_rate=1, n=10)
```

predicting y and storing in y_pred

In [27]:

```
y_pred=prediction(X_test, w=w, b=b)
```

Printing Y_intercept

In [28]:

```
print('y_intercept=',b)
```

```
y_intercept= [[6.94632157]]
```

Printing weight vector

In [29]:

```
print("weight_vector=",w)
```

```
weight_vector= [[ 2.43174227e-01 -4.67565663e+00 -8.88478932e-02 -5.13919257e-02
   7.84349763e-01 -6.58677673e-02 -4.32141826e-01 -1.58895245e+00
  -1.87077695e-02 -1.04251245e+01 -6.87462807e+00  2.51433614e+01
  -4.03347494e+00  1.68983457e+01  4.25512606e+01  2.85326219e+00
  -1.69470782e+01  1.14511432e-01  2.96478015e+00  4.66591846e+01
  -7.93868525e-01 -6.50405086e+00 -3.16617119e-01  3.85053573e+00
  -1.71406515e-01  4.75310694e-01  2.58675309e+00 -1.24818113e-01
   1.52352306e-01  2.41220237e+00]]
```

Performance metric-Accuracy

In [30]:

```
def accuracy(y_pred,y_test):
    k=list(y_test)
    y_new=[]
    for i in k:
        if i==0:
            y_new.append(-1)
        else:
            y_new.append(1)
    a=np.array(y_new)
    b=a*y_pred
    s=0
    for i in b:
        if i>0:
            s+=1
    print("model accuracy",(s/len(b))*100)
```

Accuracy for model with n_iter=10

In [31]:

```
accuracy(y_pred,y_test)
```

```
model accuracy 50.0
```

Case 2: with n_iter=100

In [32]:

```
w,b=self_implement_log_reg(self_implement_train, lr_rate=1, n=100)
```

predicting y uing obtained w & b

In [33]:

```
y_pred=prediction(X_test, w=w, b=b)
```

Printing y_intercept

In [34]:

```
print('y_intercept=',b)
```

```
y_intercept= [[311.19708205]]
```

Printing weight_vector

In [35]:

```
print("weight_vector=",w)
```

```
weight_vector= [[  1.40403189   0.69314293   0.99700111  -2.68723831  -0.38960291
   -3.00208139  -4.9134768   -1.75426428   0.08571456  15.71374931
   -0.43373416   3.8846007   -1.98900694  -1.50897404 -18.4095403
  -15.085086    19.46628819   8.70363645  10.32420065   8.55705946
    0.16904473   0.09232057   0.61508404   0.16172958   1.90614234
    4.06391336  10.94463951   4.11332915   7.86126196 -15.42975835]]
```

In [36]:

```
accuracy(y_pred,y_test)
```

```
model accuracy 63.297872340425535
```

In [40]:

```
from prettytable import PrettyTable
#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable
p = PrettyTable()
p.field_names = ["algorithm","L2 regularization:alpha","accuracy(%)","n_iterations"]
p.add_row(["inbuilt-SGD",'0.0001',94.14,10])
p.add_row(["inbuilt-SGD",'0.0001',95.21,100])
p.add_row(["self-implement-LR-SGD",'0.0001',50.00,10])
p.add_row(["self-implement-LR-SGD",'0.0001',63.29,100])
print(p)
```

```
+-----------------------+-------------------------+-------------+--------------+
|       algorithm       | L2 regularization:alpha | accuracy(%) | n_iterations |
+-----------------------+-------------------------+-------------+--------------+
|      inbuilt-SGD      |          0.0001         |    94.14    |      10      |
|      inbuilt-SGD      |          0.0001         |    95.21    |     100      |
| self-implement-LR-SGD |          0.0001         |     50.0    |      10      |
| self-implement-LR-SGD |          0.0001         |    63.29    |     100      |
+-----------------------+-------------------------+-------------+--------------+
```

Conclusion: 1.As n_iter increases accuracy increases. 2.but inbuilt classifier gives more accurate results as compared to self implemented model