In [14]:

```
import warnings
warnings.filterwarnings("ignore")
from sklearn.datasets import load_breast_cancer
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 SGDClassifier
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error
from collections import Counter
```

Loading the Boston data

In [2]:

```
bc=load_breast_cancer()
print(bc.data.shape)
```

```
(569, 30)
```

In [3]:

```
print(bc.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 [4]:

```
print(bc.target)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
```

In [5]:

```
# Description of Boston Data
print(bc.DESCR)
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 3 is Mean Radius, field
        13 is Radius SE, field 23 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
    compactness (standard error):         0.002  0.135
    concavity (standard error):           0.0    0.396
    concave points (standard error):      0.0    0.053
    symmetry (standard error):            0.008  0.079
    fractal dimension (standard error):   0.001  0.03
    radius (worst):                       7.93   36.04
    texture (worst):                      12.02  49.54
    perimeter (worst):                    50.41  251.2
    area (worst):                         185.2  4254.0
    smoothness (worst):                   0.071  0.223
    compactness (worst):                  0.027  1.058
    concavity (worst):                    0.0    1.252
    concave points (worst):               0.0    0.291
    symmetry (worst):                     0.156  0.664
    fractal dimension (worst):            0.055  0.208
    ===================================== ====== ======

    :Missing Attribute Values: None

    :Class Distribution: 212 - Malignant, 357 - Benign

    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian
```

```
    :Donor: Nick Street

    :Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. topic:: References

   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
     Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
     San Jose, CA, 1993.
   - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
     prognosis via linear programming. Operations Research, 43(4), pages 570-577,
     July-August 1995.
   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques
     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)
     163-171.
```

In [6]:

```
bc1=pd.DataFrame(bc.data)
print(bc1.head)
```

```
<bound method NDFrame.head of          0      1       2       3        4        5         6
\
0    17.990  10.38  122.80  1001.0  0.11840  0.27760  0.300100  0.147100
1    20.570  17.77  132.90  1326.0  0.08474  0.07864  0.086900  0.070170
2    19.690  21.25  130.00  1203.0  0.10960  0.15990  0.197400  0.127900
3    11.420  20.38   77.58   386.1  0.14250  0.28390  0.241400  0.105200
4    20.290  14.34  135.10  1297.0  0.10030  0.13280  0.198000  0.104300
5    12.450  15.70   82.57   477.1  0.12780  0.17000  0.157800  0.080890
6    18.250  19.98  119.60  1040.0  0.09463  0.10900  0.112700  0.074000
7    13.710  20.83   90.20   577.9  0.11890  0.16450  0.093660  0.059850
8    13.000  21.82   87.50   519.8  0.12730  0.19320  0.185900  0.093530
9    12.460  24.04   83.97   475.9  0.11860  0.23960  0.227300  0.085430
10   16.020  23.24  102.70   797.8  0.08206  0.06669  0.032990  0.033230
11   15.780  17.89  103.60   781.0  0.09710  0.12920  0.099540  0.066060
12   19.170  24.80  132.40  1123.0  0.09740  0.24580  0.206500  0.111800
13   15.850  23.95  103.70   782.7  0.08401  0.10020  0.099380  0.053640
14   13.730  22.61   93.60   578.3  0.11310  0.22930  0.212800  0.080250
15   14.540  27.54   96.73   658.8  0.11390  0.15950  0.163900  0.073640
16   14.680  20.13   94.74   684.5  0.09867  0.07200  0.073950  0.052590
17   16.130  20.68  108.10   798.8  0.11700  0.20220  0.172200  0.102800
18   19.810  22.15  130.00  1260.0  0.09831  0.10270  0.147900  0.094980
19   13.540  14.36   87.46   566.3  0.09779  0.08129  0.066640  0.047810
20   13.080  15.71   85.63   520.0  0.10750  0.12700  0.045680  0.031100
21    9.504  12.44   60.34   273.9  0.10240  0.06492  0.029560  0.020760
22   15.340  14.26  102.50   704.4  0.10730  0.21350  0.207700  0.097560
23   21.160  23.04  137.20  1404.0  0.09428  0.10220  0.109700  0.086320
24   16.650  21.38  110.00   904.6  0.11210  0.14570  0.152500  0.091700
```

24	16.650	21.38	110.00	904.6	0.11210	0.14570	0.152500	0.091700
25	17.140	16.40	116.00	912.7	0.11860	0.22760	0.222900	0.140100
26	14.580	21.53	97.41	644.8	0.10540	0.18680	0.142500	0.087830
27	18.610	20.25	122.10	1094.0	0.09440	0.10660	0.149000	0.077310
28	15.300	25.27	102.40	732.4	0.10820	0.16970	0.168300	0.087510
29	17.570	15.05	115.00	955.1	0.09847	0.11570	0.098750	0.079530
..	...	...	...	...	...	...	...	...
539	7.691	25.44	48.34	170.4	0.08668	0.11990	0.092520	0.013640
540	11.540	14.44	74.65	402.9	0.09984	0.11200	0.067370	0.025940
541	14.470	24.99	95.81	656.4	0.08837	0.12300	0.100900	0.038900
542	14.740	25.42	94.70	668.6	0.08275	0.07214	0.041050	0.030270
543	13.210	28.06	84.88	538.4	0.08671	0.06877	0.029870	0.032750
544	13.870	20.70	89.77	584.8	0.09578	0.10180	0.036880	0.023690
545	13.620	23.23	87.19	573.2	0.09246	0.06747	0.029740	0.024430
546	10.320	16.35	65.31	324.9	0.09434	0.04994	0.010120	0.005495
547	10.260	16.58	65.85	320.8	0.08877	0.08066	0.043580	0.024380
548	9.683	19.34	61.05	285.7	0.08491	0.05030	0.023370	0.009615
549	10.820	24.21	68.89	361.6	0.08192	0.06602	0.015480	0.008160
550	10.860	21.48	68.51	360.5	0.07431	0.04227	0.000000	0.000000
551	11.130	22.44	71.49	378.4	0.09566	0.08194	0.048240	0.022570
552	12.770	29.43	81.35	507.9	0.08276	0.04234	0.019970	0.014990
553	9.333	21.94	59.01	264.0	0.09240	0.05605	0.039960	0.012820
554	12.880	28.92	82.50	514.3	0.08123	0.05824	0.061950	0.023430
555	10.290	27.61	65.67	321.4	0.09030	0.07658	0.059990	0.027380
556	10.160	19.59	64.73	311.7	0.10030	0.07504	0.005025	0.011160
557	9.423	27.88	59.26	271.3	0.08123	0.04971	0.000000	0.000000
558	14.590	22.68	96.39	657.1	0.08473	0.13300	0.102900	0.037360
559	11.510	23.93	74.52	403.5	0.09261	0.10210	0.111200	0.041050
560	14.050	27.15	91.38	600.4	0.09929	0.11260	0.044620	0.043040
561	11.200	29.37	70.67	386.0	0.07449	0.03558	0.000000	0.000000
562	15.220	30.62	103.40	716.9	0.10480	0.20870	0.255000	0.094290
563	20.920	25.09	143.00	1347.0	0.10990	0.22360	0.317400	0.147400
564	21.560	22.39	142.00	1479.0	0.11100	0.11590	0.243900	0.138900
565	20.130	28.25	131.20	1261.0	0.09780	0.10340	0.144000	0.097910
566	16.600	28.08	108.30	858.1	0.08455	0.10230	0.092510	0.053020
567	20.600	29.33	140.10	1265.0	0.11780	0.27700	0.351400	0.152000
568	7.760	24.54	47.92	181.0	0.05263	0.04362	0.000000	0.000000

	8	9	...	20	21	22	23	24	25	\
0	0.2419	0.07871	...	25.380	17.33	184.60	2019.0	0.16220	0.66560	
1	0.1812	0.05667	...	24.990	23.41	158.80	1956.0	0.12380	0.18660	
2	0.2069	0.05999	...	23.570	25.53	152.50	1709.0	0.14440	0.42450	
3	0.2597	0.09744	...	14.910	26.50	98.87	567.7	0.20980	0.86630	
4	0.1809	0.05883	...	22.540	16.67	152.20	1575.0	0.13740	0.20500	
5	0.2087	0.07613	...	15.470	23.75	103.40	741.6	0.17910	0.52490	
6	0.1794	0.05742	...	22.880	27.66	153.20	1606.0	0.14420	0.25760	
7	0.2196	0.07451	...	17.060	28.14	110.60	897.0	0.16540	0.36820	
8	0.2350	0.07389	...	15.490	30.73	106.20	739.3	0.17030	0.54010	
9	0.2030	0.08243	...	15.090	40.68	97.65	711.4	0.18530	1.05800	
10	0.1528	0.05697	...	19.190	33.88	123.80	1150.0	0.11810	0.15510	
11	0.1842	0.06082	...	20.420	27.28	136.50	1299.0	0.13960	0.56090	
12	0.2397	0.07800	...	20.960	29.94	151.70	1332.0	0.10370	0.39030	
13	0.1847	0.05338	...	16.840	27.66	112.00	876.5	0.11310	0.19240	
14	0.2069	0.07682	...	15.030	32.01	108.80	697.7	0.16510	0.77250	
15	0.2303	0.07077	...	17.460	37.13	124.10	943.2	0.16780	0.65770	
16	0.1586	0.05922	...	19.070	30.88	123.40	1138.0	0.14640	0.18710	
17	0.2164	0.07356	...	20.960	31.48	136.80	1315.0	0.17890	0.42330	
18	0.1582	0.05395	...	27.320	30.88	186.80	2398.0	0.15120	0.31500	
19	0.1885	0.05766	...	15.110	19.26	99.70	711.2	0.14400	0.17730	
20	0.1967	0.06811	...	14.500	20.49	96.09	630.5	0.13120	0.27760	
21	0.1815	0.06905	...	10.230	15.66	65.13	314.9	0.13240	0.11480	
22	0.2521	0.07032	...	18.070	19.08	125.10	980.9	0.13900	0.59540	
23	0.1769	0.05278	...	29.170	35.59	188.00	2615.0	0.14010	0.26000	
24	0.1995	0.06330	...	26.460	31.56	177.00	2215.0	0.18050	0.35780	
25	0.3040	0.07413	...	22.250	21.40	152.40	1461.0	0.15450	0.39490	
26	0.2252	0.06924	...	17.620	33.21	122.40	896.9	0.15250	0.66430	
27	0.1697	0.05699	...	21.310	27.26	139.90	1403.0	0.13380	0.21170	
28	0.1926	0.06540	...	20.270	36.71	149.30	1269.0	0.16410	0.61100	
29	0.1739	0.06149	...	20.010	19.52	134.90	1227.0	0.12550	0.28120	
..	...	...	...	...	...	...	...	...	...	
539	0.2037	0.07751	...	8.678	31.89	54.49	223.6	0.15960	0.30640	
540	0.1818	0.06782	...	12.260	19.68	78.78	457.8	0.13450	0.21180	
541	0.1872	0.06341	...	16.220	31.73	113.50	808.9	0.13400	0.42020	
542	0.1840	0.05680	...	16.510	32.29	107.40	826.4	0.10600	0.13760	
543	0.1628	0.05781	...	14.370	37.17	92.48	629.6	0.10720	0.13810	
544	0.1620	0.06688	...	15.050	24.75	99.17	688.6	0.12640	0.20370	
545	0.1664	0.05801	...	15.350	29.09	97.58	729.8	0.12160	0.15170	

546	0.1885	0.06201	...	11.250	21.77	71.12	384.9	0.12850	0.08842
547	0.1669	0.06714	...	10.830	22.04	71.08	357.4	0.14610	0.22460
548	0.1580	0.06235	...	10.930	25.59	69.10	364.2	0.11990	0.09546
549	0.1976	0.06328	...	13.030	31.45	83.90	505.6	0.12040	0.16330
550	0.1661	0.05948	...	11.660	24.77	74.08	412.3	0.10010	0.07348
551	0.2030	0.06552	...	12.020	28.26	77.80	436.6	0.10870	0.17820
552	0.1539	0.05637	...	13.870	36.00	88.10	594.7	0.12340	0.10640
553	0.1692	0.06576	...	9.845	25.05	62.86	295.8	0.11030	0.08298
554	0.1566	0.05708	...	13.890	35.74	88.84	595.7	0.12270	0.16200
555	0.1593	0.06127	...	10.840	34.91	69.57	357.6	0.13840	0.17100
556	0.1791	0.06331	...	10.650	22.88	67.88	347.3	0.12650	0.12000
557	0.1742	0.06059	...	10.490	34.24	66.50	330.6	0.10730	0.07158
558	0.1454	0.06147	...	15.480	27.27	105.90	733.5	0.10260	0.31710
559	0.1388	0.06570	...	12.480	37.16	82.28	474.2	0.12980	0.25170
560	0.1537	0.06171	...	15.300	33.17	100.20	706.7	0.12410	0.22640
561	0.1060	0.05502	...	11.920	38.30	75.19	439.6	0.09267	0.05494
562	0.2128	0.07152	...	17.520	42.79	128.70	915.0	0.14170	0.79170
563	0.2149	0.06879	...	24.290	29.41	179.10	1819.0	0.14070	0.41860
564	0.1726	0.05623	...	25.450	26.40	166.10	2027.0	0.14100	0.21130
565	0.1752	0.05533	...	23.690	38.25	155.00	1731.0	0.11660	0.19220
566	0.1590	0.05648	...	18.980	34.12	126.70	1124.0	0.11390	0.30940
567	0.2397	0.07016	...	25.740	39.42	184.60	1821.0	0.16500	0.86810
568	0.1587	0.05884	...	9.456	30.37	59.16	268.6	0.08996	0.06444

	26	27	28	29
0	0.71190	0.26540	0.4601	0.11890
1	0.24160	0.18600	0.2750	0.08902
2	0.45040	0.24300	0.3613	0.08758
3	0.68690	0.25750	0.6638	0.17300
4	0.40000	0.16250	0.2364	0.07678
5	0.53550	0.17410	0.3985	0.12440
6	0.37840	0.19320	0.3063	0.08368
7	0.26780	0.15560	0.3196	0.11510
8	0.53900	0.20600	0.4378	0.10720
9	1.10500	0.22100	0.4366	0.20750
10	0.14590	0.09975	0.2948	0.08452
11	0.39650	0.18100	0.3792	0.10480
12	0.36390	0.17670	0.3176	0.10230
13	0.23220	0.11190	0.2809	0.06287
14	0.69430	0.22080	0.3596	0.14310
15	0.70260	0.17120	0.4218	0.13410
16	0.29140	0.16090	0.3029	0.08216
17	0.47840	0.20730	0.3706	0.11420
18	0.53720	0.23880	0.2768	0.07615
19	0.23900	0.12880	0.2977	0.07259
20	0.18900	0.07283	0.3184	0.08183
21	0.08867	0.06227	0.2450	0.07773
22	0.63050	0.23930	0.4667	0.09946
23	0.31550	0.20090	0.2822	0.07526
24	0.46950	0.20950	0.3613	0.09564
25	0.38530	0.25500	0.4066	0.10590
26	0.55390	0.27010	0.4264	0.12750
27	0.34460	0.14900	0.2341	0.07421
28	0.63350	0.20240	0.4027	0.09876
29	0.24890	0.14560	0.2756	0.07919
..	...	...	...	...
539	0.33930	0.05000	0.2790	0.10660
540	0.17970	0.06918	0.2329	0.08134
541	0.40400	0.12050	0.3187	0.10230
542	0.16110	0.10950	0.2722	0.06956
543	0.10620	0.07958	0.2473	0.06443
544	0.13770	0.06845	0.2249	0.08492
545	0.10490	0.07174	0.2642	0.06953
546	0.04384	0.02381	0.2681	0.07399
547	0.17830	0.08333	0.2691	0.09479
548	0.09350	0.03846	0.2552	0.07920
549	0.06194	0.03264	0.3059	0.07626
550	0.00000	0.00000	0.2458	0.06592
551	0.15640	0.06413	0.3169	0.08032
552	0.08653	0.06498	0.2407	0.06484
553	0.07993	0.02564	0.2435	0.07393
554	0.24390	0.06493	0.2372	0.07242
555	0.20000	0.09127	0.2226	0.08283
556	0.01005	0.02232	0.2262	0.06742
557	0.00000	0.00000	0.2475	0.06969
558	0.36620	0.11050	0.2258	0.08004
559	0.36300	0.09653	0.2112	0.08732

```
560   0.13260   0.10480   0.2250   0.08321
561   0.00000   0.00000   0.1566   0.05905
562   1.17000   0.23560   0.4089   0.14090
563   0.65990   0.25420   0.2929   0.09873
564   0.41070   0.22160   0.2060   0.07115
565   0.32150   0.16280   0.2572   0.06637
566   0.34030   0.14180   0.2218   0.07820
567   0.93870   0.26500   0.4087   0.12400
568   0.00000   0.00000   0.2871   0.07039

[569 rows x 30 columns]>
```

In [7]:

```
bc1['Detect']=bc.target
x=bc1.drop('Detect',axis=1)
y=bc1['Detect']
print(x.shape)
print(y.shape)
```

```
(569, 30)
(569,)
```

Splitting the Data into train and test set

In [8]:

```
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)
```

In [9]:

```
print("shape of train data ")
print(x_train.shape)
print(y_train.shape)
print("shape of test data ")
print(x_test.shape)
print(y_test.shape)
```

```
shape of train data
(398, 30)
(398,)
shape of test data
(171, 30)
(171,)
```

In [10]:

```
train_data=pd.concat([x_train,y_train],axis=1)
train_data.shape
```

Out[10]:

```
(398, 31)
```

In [11]:

```
test_data=pd.concat([x_test,y_test],axis=1)
test_data.shape
```

Out[11]:

```
(171, 31)
```

In [12]:

```
xtrain1=np.array(x_train)
ytrain1=np.array(y_train)
```

```
xtest1=np.array(x_test)
ytest1=np.array(y_test)
```

In [15]:

```
print("-"*10, "Distribution of output variable in train data", "-"*10)
train_distr = Counter(y_train)
train_len = len(y_train)
print("Class 0: ",int(train_distr[0])/train_len,"Class 1: ", int(train_distr[1])/train_len)
print("-"*10, "Distribution of output variable in test data", "-"*10)
test_distr = Counter(y_test)

test_len = len(y_test)
print("Class 0: ",int(test_distr[0])/test_len, "Class 1: ",int(test_distr[1])/test_len)
```

```
---------- Distribution of output variable in train data ----------
Class 0:  0.3768844221105528 Class 1:  0.6231155778894473
---------- Distribution of output variable in test data ----------
Class 0:  0.36257309941520466 Class 1:  0.6374269005847953
```

Custom SGD Implementation

In [ ]:

```
```

In [16]:

```
def sgdcustom(x_train,y_train,train,learning_rate=1,n_itr=2000,k=10,Lambda=1):
    w_cur=np.zeros(shape=(1,x_train.shape[1]))
    b_cur=0
    cur_itr=0
    cur_cost=10000000

    curcost=[]
    flag=0
    print("Learning Rate ",learning_rate)
    #cur_cost=
[10000000,10000000,10000000,10000000,10000000,10000000,10000000,10000000,10000000,10000000,1000000(
00000,10000000]
    while(cur_itr<=n_itr):
        #plt.plot(cur_itr,cur_cost)
        #plt.show()
        #print(cur_cost)
        curcost.append(cur_cost)
        w_old=w_cur
        b_old=b_cur
        w_temp=np.zeros(shape=(1,x_train.shape[1]))
        b_temp=0
        temp=train_data.sample(k)
        #print(temp.head(3))
        y=np.array(temp['Detect'])
        x=np.array(temp.drop('Detect',axis=1))
        error=np.zeros(shape=(1,x_train.shape[1]))
        for i in range(k):
            y_hat=np.asscalar(1.0/(1.0+np.exp(-(np.dot(w_old,x[i])+b_old))))
            w_temp+=x[i]*(y[i]-y_hat)*(-2/k)*(y_hat)*(1-y_hat)
            b_temp+=(y[i]-y_hat)*(-2/k)*y_hat*(1-y_hat)
            error+= (-y[i] * np.log(y_hat)) -((1-y[i])*np.log(1-y_hat))



            cost=(1/k)*sum(sum(error))

            a=((Lambda/(2*k))*sum(sum(w_old**2)))
        #print(a)
        #print(error)
            regCost= cost + (Lambda/(2*k)) * sum(sum(w_old**2))
            #print(regCost)
            w_cur=w_old-learning_rate*w_temp
            b_cur=b_old-learning_rate*b_temp
```

```
            if(np.isnan(regCost)==False):
                if ((cur_cost-regCost)<0.00001):
                    if(regCost<cur_cost):
                        cur_cost=regCost
                        flag=1
                        break
                else:
                    cur_cost=regCost


        cur_itr+=1
        #if(np.absolute(w_old-w_cur)<=0.00001).all():
        if flag==1 :
            break
        # break

        learning_rate=learning_rate/pow(n_itr*k,learning_rate)
    itr=list(range(0,cur_itr))
    plt.plot(itr,curcost)
    plt.xlabel('Iterations')
    plt.ylabel('Cost')
    plt.show()

    return w_cur,b_cur
def predict(x,w,b):
    ypred=[]
    for i in range(0,len(x)):
        y=np.asscalar(np.dot(w,x[i])+b)
        y=np.asscalar(1.0/(1.0+np.exp(-y)))
        ypred.append(y)
    return np.array(ypred)
```

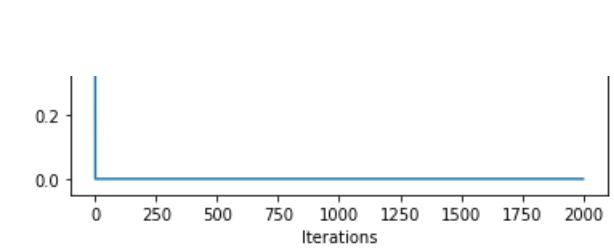
In [20]:

```
# Plotting actual data vs predicted data
def plot_(test_data,y_pred):
    #scatter plot
    plt.scatter(test_data,y_pred)

    plt.title('scatter plot of actual y and predicted y')
    plt.xlabel('actual y')
    plt.ylabel('predicted y')
    plt.show()
```

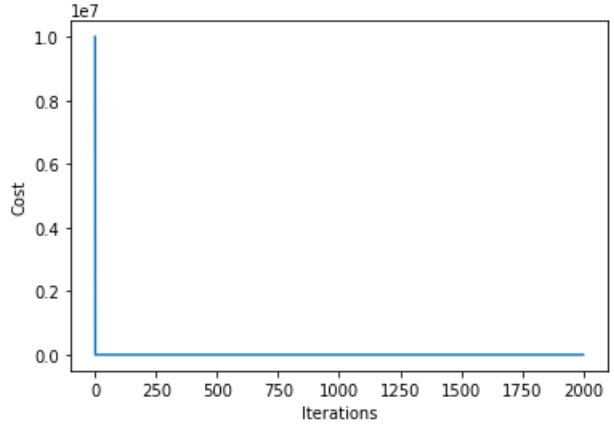
In [21]:

```
# Finding the optimal learning rate

from sklearn.metrics import log_loss
train_error=[]
learning_rate=[10,5,3,1,0.0001,0.00005,0.0000625,0.00000411,0.00000418]
for i in learning_rate:
    w1,b1=sgdcustom(x_train,y_train,train_data,learning_rate=i,Lambda=1)
    # print(w.shape,b.shape,x1_train_.shape)
    y1_pred_train=predict(xtrain1,w1,b1)
    train_error.append(log_loss(ytrain1,y1_pred_train))
print("Train Error",train_error)
print("minimum Train Error",min(train_error))
optimal_rate = learning_rate[train_error.index(min(train_error))]
print(optimal_rate)
```

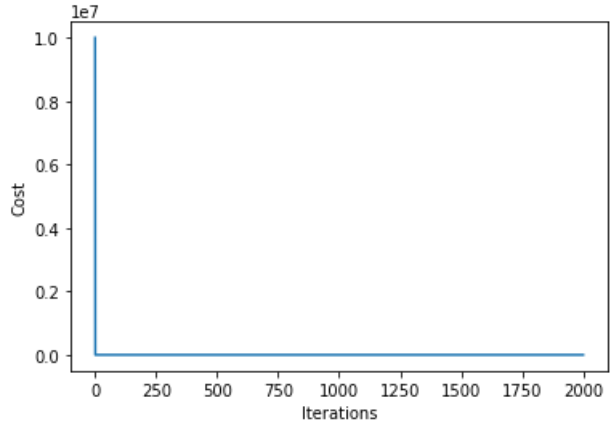
```
Learning Rate  10
```



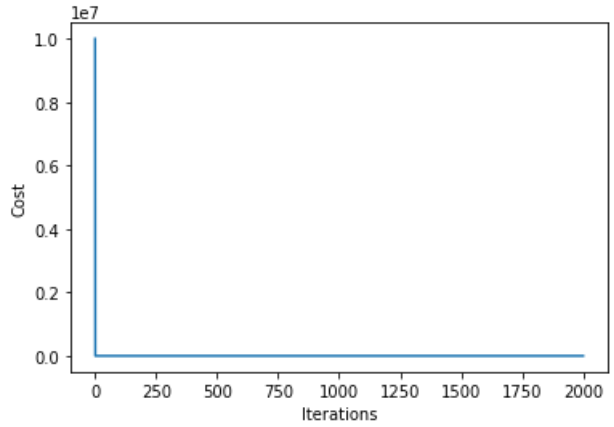
Learning Rate 5



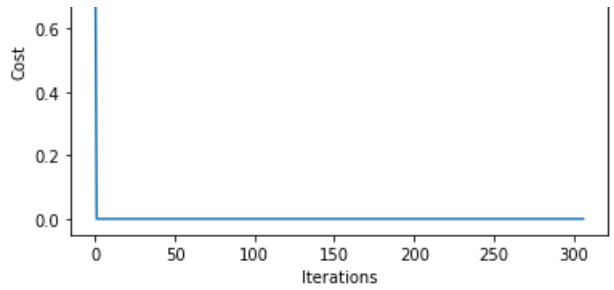
Learning Rate 3



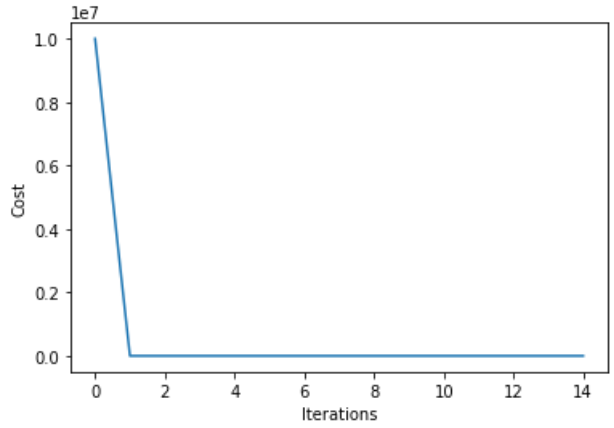
Learning Rate 1



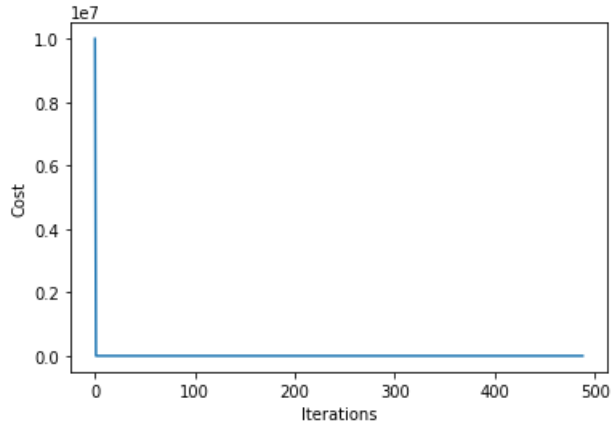
Learning Rate 0.0001



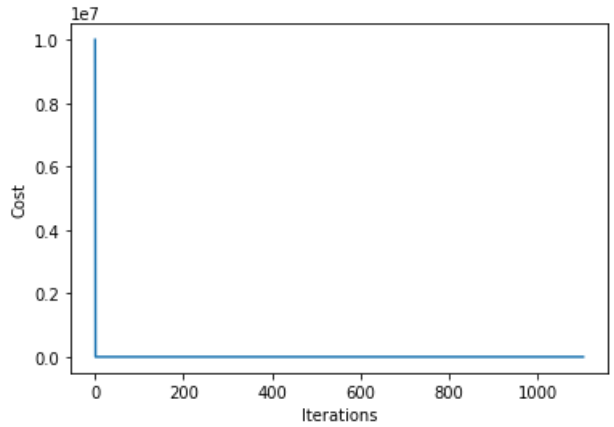
Learning Rate 5e-05



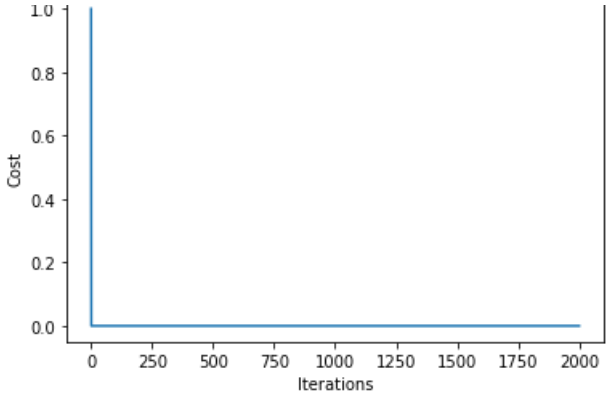
Learning Rate 6.25e-05



Learning Rate 4.11e-06



Learning Rate 4.18e-06

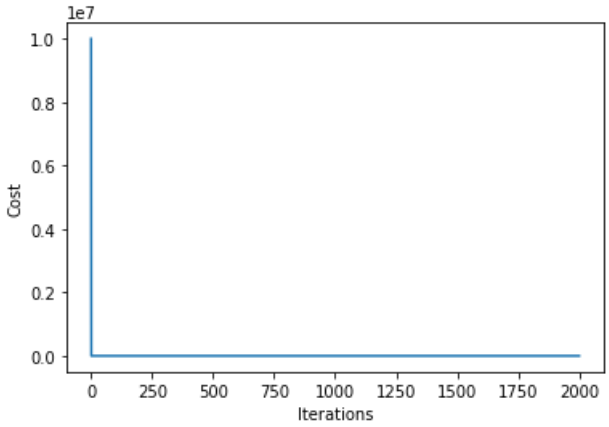



```
Train Error [13.017428137816912, 21.52164961290917, 21.52164961290917, 21.52164961290917,
4.9337081650034325, 2.2228238608069946, 5.342011711244495, 0.38573678712665765,
0.34224070142537677]
minimum Train Error 0.34224070142537677
4.18e-06
```

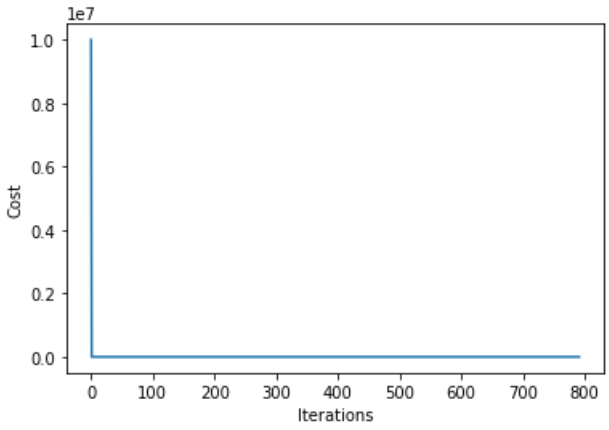
In [28]:

```
# Applying the model on test data
w,b=sgdcustom(x_train,y_train,train_data,learning_rate=optimal_rate)
w1,b1=sgdcustom(x_test,y_test,test_data,learning_rate=optimal_rate)
# print(w.shape,b.shape,x1_train_.shape)
y1_pred_train=predict(xtrain1,w,b)
y1_pred_test=predict(xtest1,w,b)
```

```
Learning Rate  4.18e-06
```



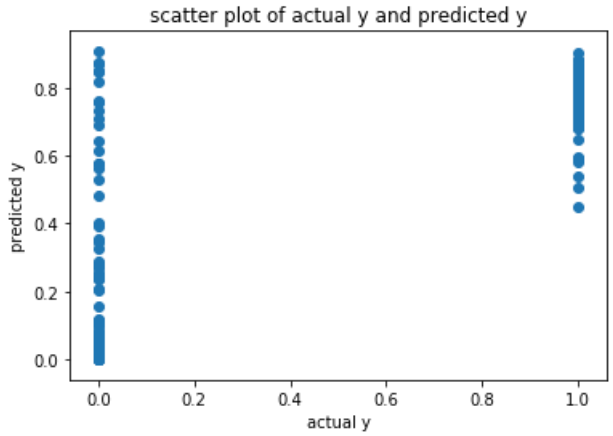

```
Learning Rate  4.18e-06
```



In [29]:

```
# plotting actual test values vs predicted test values
```

```
plot_(ytest1,y1_pred_test)
```



In [30]:

```
log_loss(ytest1,y1_pred_test)
```

Out[30]:

```
0.34625809527106677
```

In [31]:

```
from sklearn.metrics import roc_auc_score
train_acc=  roc_auc_score(ytrain1, y1_pred_train)
test_acc= roc_auc_score(ytest1, y1_pred_test)
print(test_acc)
```

```
0.8919798757028707
```

In [32]:

```
bc1=pd.DataFrame(bc.data)
print(bc1.head)
bc1['PRICE']=bc.target
x=bc1.drop('PRICE',axis=1)
y=bc1['PRICE']
print(x.shape)
print(y.shape)
```

```
<bound method NDFrame.head of           0      1       2       3        4        5         6
\
0    17.990  10.38  122.80  1001.0  0.11840  0.27760  0.300100  0.147100
1    20.570  17.77  132.90  1326.0  0.08474  0.07864  0.086900  0.070170
2    19.690  21.25  130.00  1203.0  0.10960  0.15990  0.197400  0.127900
3    11.420  20.38   77.58   386.1  0.14250  0.28390  0.241400  0.105200
4    20.290  14.34  135.10  1297.0  0.10030  0.13280  0.198000  0.104300
5    12.450  15.70   82.57   477.1  0.12780  0.17000  0.157800  0.080890
6    18.250  19.98  119.60  1040.0  0.09463  0.10900  0.112700  0.074000
7    13.710  20.83   90.20   577.9  0.11890  0.16450  0.093660  0.059850
8    13.000  21.82   87.50   519.8  0.12730  0.19320  0.185900  0.093530
9    12.460  24.04   83.97   475.9  0.11860  0.23960  0.227300  0.085430
10   16.020  23.24  102.70   797.8  0.08206  0.06669  0.032990  0.033230
11   15.780  17.89  103.60   781.0  0.09710  0.12920  0.099540  0.066060
12   19.170  24.80  132.40  1123.0  0.09740  0.24580  0.206500  0.111800
13   15.850  23.95  103.70   782.7  0.08401  0.10020  0.099380  0.053640
14   13.730  22.61   93.60   578.3  0.11310  0.22930  0.212800  0.080250
15   14.540  27.54   96.73   658.8  0.11390  0.15950  0.163900  0.073640
16   14.680  20.13   94.74   684.5  0.09867  0.07200  0.073950  0.052590
17   16.130  20.68  108.10   798.8  0.11700  0.20220  0.172200  0.102800
18   19.810  22.15  130.00  1260.0  0.09831  0.10270  0.147900  0.094980
19   13.540  14.36   87.46   566.3  0.09779  0.08129  0.066640  0.047810
20   13.080  15.71   85.63   520.0  0.10750  0.12700  0.045680  0.031100
21    9.504  12.44   60.34   273.9  0.10240  0.06492  0.029560  0.020760
22   15.340  14.26  102.50   704.4  0.10730  0.21350  0.207700  0.097560
```

23	21.160	23.04	137.20	1404.0	0.09428	0.10220	0.109700	0.086320	
24	16.650	21.38	110.00	904.6	0.11210	0.14570	0.152500	0.091700	
25	17.140	16.40	116.00	912.7	0.11860	0.22760	0.222900	0.140100	
26	14.580	21.53	97.41	644.8	0.10540	0.18680	0.142500	0.087830	
27	18.610	20.25	122.10	1094.0	0.09440	0.10660	0.149000	0.077310	
28	15.300	25.27	102.40	732.4	0.10820	0.16970	0.168300	0.087510	
29	17.570	15.05	115.00	955.1	0.09847	0.11570	0.098750	0.079530	
..	...	...	...	...	...	...	...	...	
539	7.691	25.44	48.34	170.4	0.08668	0.11990	0.092520	0.013640	
540	11.540	14.44	74.65	402.9	0.09984	0.11200	0.067370	0.025940	
541	14.470	24.99	95.81	656.4	0.08837	0.12300	0.100900	0.038900	
542	14.740	25.42	94.70	668.6	0.08275	0.07214	0.041050	0.030270	
543	13.210	28.06	84.88	538.4	0.08671	0.06877	0.029870	0.032750	
544	13.870	20.70	89.77	584.8	0.09578	0.10180	0.036880	0.023690	
545	13.620	23.23	87.19	573.2	0.09246	0.06747	0.029740	0.024430	
546	10.320	16.35	65.31	324.9	0.09434	0.04994	0.010120	0.005495	
547	10.260	16.58	65.85	320.8	0.08877	0.08066	0.043580	0.024380	
548	9.683	19.34	61.05	285.7	0.08491	0.05030	0.023370	0.009615	
549	10.820	24.21	68.89	361.6	0.08192	0.06602	0.015480	0.008160	
550	10.860	21.48	68.51	360.5	0.07431	0.04227	0.000000	0.000000	
551	11.130	22.44	71.49	378.4	0.09566	0.08194	0.048240	0.022570	
552	12.770	29.43	81.35	507.9	0.08276	0.04234	0.019970	0.014990	
553	9.333	21.94	59.01	264.0	0.09240	0.05605	0.039960	0.012820	
554	12.880	28.92	82.50	514.3	0.08123	0.05824	0.061950	0.023430	
555	10.290	27.61	65.67	321.4	0.09030	0.07658	0.059990	0.027380	
556	10.160	19.59	64.73	311.7	0.10030	0.07504	0.005025	0.011160	
557	9.423	27.88	59.26	271.3	0.08123	0.04971	0.000000	0.000000	
558	14.590	22.68	96.39	657.1	0.08473	0.13300	0.102900	0.037360	
559	11.510	23.93	74.52	403.5	0.09261	0.10210	0.111200	0.041050	
560	14.050	27.15	91.38	600.4	0.09929	0.11260	0.044620	0.043040	
561	11.200	29.37	70.67	386.0	0.07449	0.03558	0.000000	0.000000	
562	15.220	30.62	103.40	716.9	0.10480	0.20870	0.255000	0.094290	
563	20.920	25.09	143.00	1347.0	0.10990	0.22360	0.317400	0.147400	
564	21.560	22.39	142.00	1479.0	0.11100	0.11590	0.243900	0.138900	
565	20.130	28.25	131.20	1261.0	0.09780	0.10340	0.144000	0.097910	
566	16.600	28.08	108.30	858.1	0.08455	0.10230	0.092510	0.053020	
567	20.600	29.33	140.10	1265.0	0.11780	0.27700	0.351400	0.152000	
568	7.760	24.54	47.92	181.0	0.05263	0.04362	0.000000	0.000000	

	8	9	...	20	21	22	23	24	25	\
0	0.2419	0.07871	...	25.380	17.33	184.60	2019.0	0.16220	0.66560	
1	0.1812	0.05667	...	24.990	23.41	158.80	1956.0	0.12380	0.18660	
2	0.2069	0.05999	...	23.570	25.53	152.50	1709.0	0.14440	0.42450	
3	0.2597	0.09744	...	14.910	26.50	98.87	567.7	0.20980	0.86630	
4	0.1809	0.05883	...	22.540	16.67	152.20	1575.0	0.13740	0.20500	
5	0.2087	0.07613	...	15.470	23.75	103.40	741.6	0.17910	0.52490	
6	0.1794	0.05742	...	22.880	27.66	153.20	1606.0	0.14420	0.25760	
7	0.2196	0.07451	...	17.060	28.14	110.60	897.0	0.16540	0.36820	
8	0.2350	0.07389	...	15.490	30.73	106.20	739.3	0.17030	0.54010	
9	0.2030	0.08243	...	15.090	40.68	97.65	711.4	0.18530	1.05800	
10	0.1528	0.05697	...	19.190	33.88	123.80	1150.0	0.11810	0.15510	
11	0.1842	0.06082	...	20.420	27.28	136.50	1299.0	0.13960	0.56090	
12	0.2397	0.07800	...	20.960	29.94	151.70	1332.0	0.10370	0.39030	
13	0.1847	0.05338	...	16.840	27.66	112.00	876.5	0.11310	0.19240	
14	0.2069	0.07682	...	15.030	32.01	108.80	697.7	0.16510	0.77250	
15	0.2303	0.07077	...	17.460	37.13	124.10	943.2	0.16780	0.65770	
16	0.1586	0.05922	...	19.070	30.88	123.40	1138.0	0.14640	0.18710	
17	0.2164	0.07356	...	20.960	31.48	136.80	1315.0	0.17890	0.42330	
18	0.1582	0.05395	...	27.320	30.88	186.80	2398.0	0.15120	0.31500	
19	0.1885	0.05766	...	15.110	19.26	99.70	711.2	0.14400	0.17730	
20	0.1967	0.06811	...	14.500	20.49	96.09	630.5	0.13120	0.27760	
21	0.1815	0.06905	...	10.230	15.66	65.13	314.9	0.13240	0.11480	
22	0.2521	0.07032	...	18.070	19.08	125.10	980.9	0.13900	0.59540	
23	0.1769	0.05278	...	29.170	35.59	188.00	2615.0	0.14010	0.26000	
24	0.1995	0.06330	...	26.460	31.56	177.00	2215.0	0.18050	0.35780	
25	0.3040	0.07413	...	22.250	21.40	152.40	1461.0	0.15450	0.39490	
26	0.2252	0.06924	...	17.620	33.21	122.40	896.9	0.15250	0.66430	
27	0.1697	0.05699	...	21.310	27.26	139.90	1403.0	0.13380	0.21170	
28	0.1926	0.06540	...	20.270	36.71	149.30	1269.0	0.16410	0.61100	
29	0.1739	0.06149	...	20.010	19.52	134.90	1227.0	0.12550	0.28120	
..	...	...	...	...	...	...	...	...	...	
539	0.2037	0.07751	...	8.678	31.89	54.49	223.6	0.15960	0.30640	
540	0.1818	0.06782	...	12.260	19.68	78.78	457.8	0.13450	0.21180	
541	0.1872	0.06341	...	16.220	31.73	113.50	808.9	0.13400	0.42020	
542	0.1840	0.05680	...	16.510	32.29	107.40	826.4	0.10600	0.13760	
543	0.1628	0.05781	...	14.370	37.17	92.48	629.6	0.10720	0.13810	
544	0.1620	0.06688	...	15.050	24.75	99.17	688.6	0.12640	0.20370	

545	0.1664	0.05801	...	15.350	29.09	97.58	729.8	0.12160	0.15170
546	0.1885	0.06201	...	11.250	21.77	71.12	384.9	0.12850	0.08842
547	0.1669	0.06714	...	10.830	22.04	71.08	357.4	0.14610	0.22460
548	0.1580	0.06235	...	10.930	25.59	69.10	364.2	0.11990	0.09546
549	0.1976	0.06328	...	13.030	31.45	83.90	505.6	0.12040	0.16330
550	0.1661	0.05948	...	11.660	24.77	74.08	412.3	0.10010	0.07348
551	0.2030	0.06552	...	12.020	28.26	77.80	436.6	0.10870	0.17820
552	0.1539	0.05637	...	13.870	36.00	88.10	594.7	0.12340	0.10640
553	0.1692	0.06576	...	9.845	25.05	62.86	295.8	0.11030	0.08298
554	0.1566	0.05708	...	13.890	35.74	88.84	595.7	0.12270	0.16200
555	0.1593	0.06127	...	10.840	34.91	69.57	357.6	0.13840	0.17100
556	0.1791	0.06331	...	10.650	22.88	67.88	347.3	0.12650	0.12000
557	0.1742	0.06059	...	10.490	34.24	66.50	330.6	0.10730	0.07158
558	0.1454	0.06147	...	15.480	27.27	105.90	733.5	0.10260	0.31710
559	0.1388	0.06570	...	12.480	37.16	82.28	474.2	0.12980	0.25170
560	0.1537	0.06171	...	15.300	33.17	100.20	706.7	0.12410	0.22640
561	0.1060	0.05502	...	11.920	38.30	75.19	439.6	0.09267	0.05494
562	0.2128	0.07152	...	17.520	42.79	128.70	915.0	0.14170	0.79170
563	0.2149	0.06879	...	24.290	29.41	179.10	1819.0	0.14070	0.41860
564	0.1726	0.05623	...	25.450	26.40	166.10	2027.0	0.14100	0.21130
565	0.1752	0.05533	...	23.690	38.25	155.00	1731.0	0.11660	0.19220
566	0.1590	0.05648	...	18.980	34.12	126.70	1124.0	0.11390	0.30940
567	0.2397	0.07016	...	25.740	39.42	184.60	1821.0	0.16500	0.86810
568	0.1587	0.05884	...	9.456	30.37	59.16	268.6	0.08996	0.06444

	26	27	28	29
0	0.71190	0.26540	0.4601	0.11890
1	0.24160	0.18600	0.2750	0.08902
2	0.45040	0.24300	0.3613	0.08758
3	0.68690	0.25750	0.6638	0.17300
4	0.40000	0.16250	0.2364	0.07678
5	0.53550	0.17410	0.3985	0.12440
6	0.37840	0.19320	0.3063	0.08368
7	0.26780	0.15560	0.3196	0.11510
8	0.53900	0.20600	0.4378	0.10720
9	1.10500	0.22100	0.4366	0.20750
10	0.14590	0.09975	0.2948	0.08452
11	0.39650	0.18100	0.3792	0.10480
12	0.36390	0.17670	0.3176	0.10230
13	0.23220	0.11190	0.2809	0.06287
14	0.69430	0.22080	0.3596	0.14310
15	0.70260	0.17120	0.4218	0.13410
16	0.29140	0.16090	0.3029	0.08216
17	0.47840	0.20730	0.3706	0.11420
18	0.53720	0.23880	0.2768	0.07615
19	0.23900	0.12880	0.2977	0.07259
20	0.18900	0.07283	0.3184	0.08183
21	0.08867	0.06227	0.2450	0.07773
22	0.63050	0.23930	0.4667	0.09946
23	0.31550	0.20090	0.2822	0.07526
24	0.46950	0.20950	0.3613	0.09564
25	0.38530	0.25500	0.4066	0.10590
26	0.55390	0.27010	0.4264	0.12750
27	0.34460	0.14900	0.2341	0.07421
28	0.63350	0.20240	0.4027	0.09876
29	0.24890	0.14560	0.2756	0.07919
..	...	...	...	...
539	0.33930	0.05000	0.2790	0.10660
540	0.17970	0.06918	0.2329	0.08134
541	0.40400	0.12050	0.3187	0.10230
542	0.16110	0.10950	0.2722	0.06956
543	0.10620	0.07958	0.2473	0.06443
544	0.13770	0.06845	0.2249	0.08492
545	0.10490	0.07174	0.2642	0.06953
546	0.04384	0.02381	0.2681	0.07399
547	0.17830	0.08333	0.2691	0.09479
548	0.09350	0.03846	0.2552	0.07920
549	0.06194	0.03264	0.3059	0.07626
550	0.00000	0.00000	0.2458	0.06592
551	0.15640	0.06413	0.3169	0.08032
552	0.08653	0.06498	0.2407	0.06484
553	0.07993	0.02564	0.2435	0.07393
554	0.24390	0.06493	0.2372	0.07242
555	0.20000	0.09127	0.2226	0.08283
556	0.01005	0.02232	0.2262	0.06742
557	0.00000	0.00000	0.2475	0.06969
558	0.36620	0.11050	0.2258	0.08004

```
559   0.36300   0.09653   0.2112   0.08732
560   0.13260   0.10480   0.2250   0.08321
561   0.00000   0.00000   0.1566   0.05905
562   1.17000   0.23560   0.4089   0.14090
563   0.65990   0.25420   0.2929   0.09873
564   0.41070   0.22160   0.2060   0.07115
565   0.32150   0.16280   0.2572   0.06637
566   0.34030   0.14180   0.2218   0.07820
567   0.93870   0.26500   0.4087   0.12400
568   0.00000   0.00000   0.2871   0.07039

[569 rows x 30 columns]>
(569, 30)
(569,)
```

In [33]:

```
from sklearn.model_selection import train_test_split
x_train1,x_test1,y_train1,y_test1=train_test_split(x,y,test_size=0.3)
```

In [34]:

```
# Preprocessing the data
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(x_train1)
x_train1 = scaler.transform(x_train1)
x_test1 = scaler.transform(x_test1)
```

SKLEARN SGD implementation

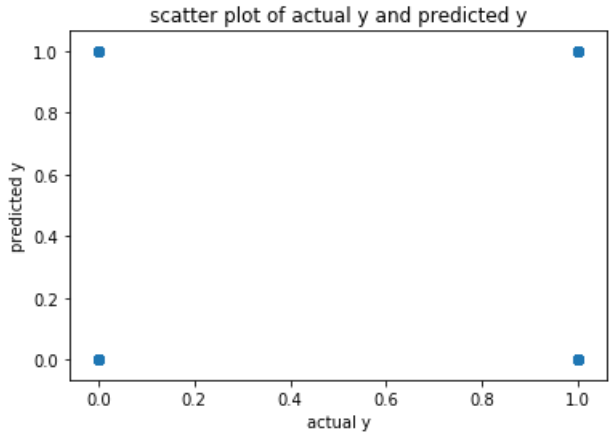
In [35]:

```
clf =  SGDClassifier(loss='log',max_iter=2000,penalty='l2',alpha=1)
clf.fit(x_train1, y_train1)
print(log_loss(y_train1, clf.predict(x_train1)))
```

```
2.0827844839731684
```

In [36]:

```
# Applying the model on test data
y1_pred_train1=clf.predict(scaler.transform(x_train1))
y1_pred_test1=clf.predict(scaler.transform(x_test1))
plot_(y_test1,y1_pred_test1)
```



In [37]:

```
train_acc_sklearn=  roc_auc_score(y_train1, y1_pred_train1)
test_acc_sklearn= roc_auc_score(y_test1, y1_pred_test1)
print(test_acc_sklearn)
```

```
0.8871065375302662
```

Comparison of weightvector and MSE

In [38]:

```
from prettytable import PrettyTable
# MSE = mean squared error
# MAE =mean absolute error
x=PrettyTable()
x.field_names=['Model','Log Loss','Train ROC_AUC Score','Test ROC_AUC Score']
x.add_row(['sklearn',log_loss(y_test1,y1_pred_test1 ),train_acc_sklearn,test_acc_sklearn])
x.add_row(['custom',log_loss(ytest1,y1_pred_test),train_acc,test_acc])
print(x)
```

```
+---------+--------------------+--------------------+--------------------+
|  Model  |      Log Loss      | Train ROC_AUC Score | Test ROC_AUC Score |
+---------+--------------------+--------------------+--------------------+
| sklearn |  3.8376745537152894 |  0.9019874616513273 | 0.8871065375302662 |
|  custom | 0.34625809527106677 |  0.928252688172043  | 0.8919798757028707 |
+---------+--------------------+--------------------+--------------------+
```

Comparison between SGD predicted value and custom SGD predicted value

In [77]:

```
x=PrettyTable()
x.field_names=['SKLearn SGD predicted value','Implemented SGD predicted value']
for i in range(15):
    x.add_row([y1_pred_test1[i],y1_pred_test[i]])
print(x)
```

```
+-----------------------------+----------------------------------+
| SKLearn SGD predicted value | Implemented SGD predicted value |
+-----------------------------+----------------------------------+
|              1              |       0.000550617133655255      |
|              1              |       0.005102306955204033      |
|              1              |       0.010173887650554703      |
|              0              |      4.455435894743123e-09      |
|              1              |      0.0003914037131936326      |
|              1              |      0.0015750070931446962      |
|              0              |      0.00011368874761726363     |
|              0              |       0.00550395606950759       |
|              1              |      0.00047116192939024277     |
|              1              |      0.0002442668409408379      |
|              1              |      5.638173229769786e-05      |
|              0              |       0.000706068343297728      |
|              1              |      0.0030800614966958783      |
|              0              |       0.08214941960989178       |
|              1              |       0.001972605955857879      |
+-----------------------------+----------------------------------+
```

In [ ]:

In [ ]: