Loading Dataset

```
import sklearn.datasets
```

```
breast_cancer = sklearn.datasets.load_breast_cancer()
```

```
x = breast_cancer.data
y = breast_cancer.target
```

```
print(x)
print(x.shape)
print(y)
print(y.shape)
```

```
[[1.799e+01 1.038e+01 1.228e+02 ... 2.654e-01 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 1.418e-01 2.218e-01 7.820e-02]
 [2.060e+01 2.933e+01 1.401e+02 ... 2.650e-01 4.087e-01 1.240e-01]
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 2.871e-01 7.039e-02]]
(569, 30)
[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 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
 1 0 1 1 0 1 0 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 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 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 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 1 1 0 0 0 0 0 0 1]
(569,)
```

```
import pandas as pd
import numpy as np
data = pd.DataFrame(breast_cancer.data,columns = breast_cancer.feature_names)
```

```
data['class'] = breast_cancer.target
```

```
data.head()
```

	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	perime er
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	1.0950	0.9053	8.
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	0.5435	0.7339	3.
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	0.7456	0.7869	4.
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	0.4956	1.1560	3.
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	0.7572	0.7813	5.

```
data.describe()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	56
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	

```
data[data['class']==1].count #we can get the no of rows that has the cancer.
```

```
<bound method DataFrame.count of      mean radius  mean texture  ...  worst fractal dimension  class
19        13.540         14.36  ...                  0.07259      1
20        13.080         15.71  ...                  0.08183      1
21         9.504         12.44  ...                  0.07773      1
37        13.030         18.42  ...                  0.06169      1
46         8.196         16.84  ...                  0.07409      1
..           ...           ...  ...                      ...    ...
558       14.590         22.68  ...                  0.08004      1
559       11.510         23.93  ...                  0.08732      1
560       14.050         27.15  ...                  0.08321      1
561       11.200         29.37  ...                  0.05905      1
568        7.760         24.54  ...                  0.07039      1

[357 rows x 31 columns]>
```

```
data.groupby('class').mean() # eg to understand it will say for how many rows it has 0 as label it would display mean of all the rows
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	radius error
class											
0	17.462830	21.604906	115.365377	978.376415	0.102898	0.145188	0.160775	0.087990	0.192909	0.062680	0.609083
1	12.146524	17.914762	78.075406	462.790196	0.092478	0.080085	0.046058	0.025717	0.174186	0.062867	0.284082

```
breast_cancer.target_names
```

```
array(['malignant', 'benign'], dtype='<U9')
```

Train Test Split

```
from sklearn.model_selection import train_test_split
```

```
X = data.drop('class',axis=1)
Y = data['class']
```

```
x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size = 0.1,stratify = Y,random_state=1)
```

```
print(Y.mean(),y_train.mean(),y_test.mean())
```

```
0.6274165202108963 0.626953125 0.631578947368421
```

```
print(y_train.shape,y_test.shape)
```

```
(512,) (57,)
```

```
print(X.mean(),x_train.mean(),x_test.mean())
```

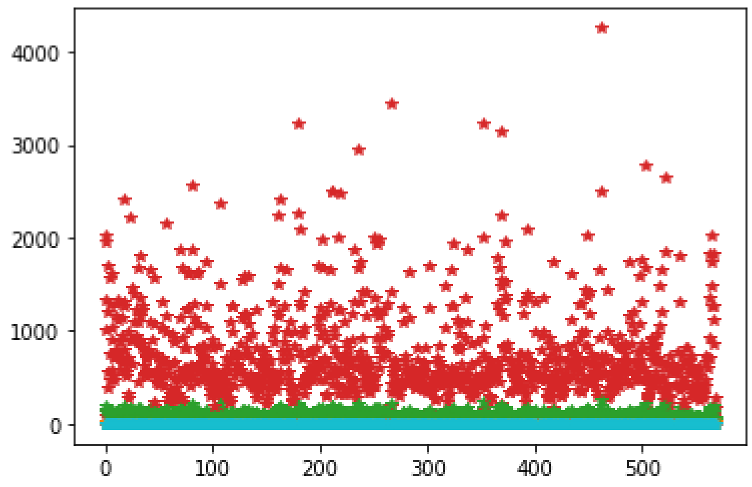
```
mean radius                 14.127292
mean texture                19.289649
mean perimeter              91.969033
mean area                  654.889104
mean smoothness              0.096360
mean compactness             0.104341
mean concavity               0.088799
mean concave points          0.048919
mean symmetry                0.181162
mean fractal dimension       0.062798
radius error                 0.405172
texture error                1.216853
perimeter error              2.866059
area error                  40.337079
smoothness error             0.007041
compactness error            0.025478
concavity error              0.031894
concave points error         0.011796
symmetry error               0.020542
fractal dimension error      0.003795
worst radius                16.269190
worst texture               25.677223
worst perimeter            107.261213
worst area                 880.583128
worst smoothness             0.132369
worst compactness            0.254265
worst concavity              0.272188
worst concave points         0.114606
worst symmetry               0.290076
worst fractal dimension      0.083946
dtype: float64 mean radius                 14.058656
mean texture                19.309668
mean perimeter              91.530488
mean area                  648.097266
mean smoothness              0.096568
mean compactness             0.105144
mean concavity               0.089342
mean concave points          0.048892
mean symmetry                0.181961
mean fractal dimension       0.062979
radius error                 0.403659
texture error                1.206856
perimeter error              2.861173
area error                  39.935506
smoothness error             0.007067
compactness error            0.025681
concavity error              0.032328
concave points error         0.011963
symmetry error               0.020584
fractal dimension error      0.003815
worst radius                16.194275
worst texture               25.644902
worst perimeter            106.757715
worst area                 871.647852
worst smoothness             0.132592
worst compactness            0.257415
worst concavity              0.275623
worst concave points         0.115454
worst symmetry               0.291562
worst fractal dimension      0.084402
dtype: float64 mean radius                 14.743807
mean texture                19.109825
mean perimeter              95.908246
mean area                  715.896491
mean smoothness              0.094496
mean compactness             0.097130
mean concavity               0.083923
mean concave points          0.049159
mean symmetry                0.173981
mean fractal dimension       0.061169
radius error                 0.418767
texture error                1.306656
perimeter error              2.909946
area error                  43.944193
smoothness error             0.006809
compactness error            0.023659
concavity error              0.027989
concave points error         0.010293
symmetry error               0.020169
fractal dimension error      0.003618
worst radius                16.942105
worst texture               25.967544
worst perimeter            111.783860
```

Binarizing of input

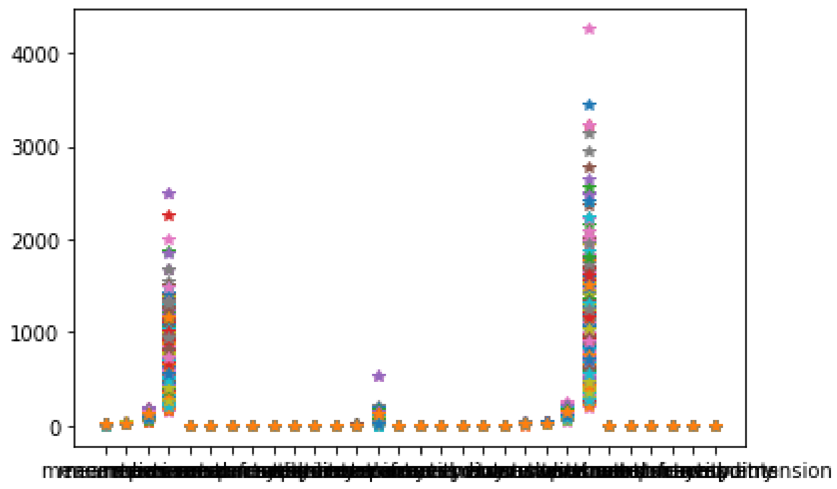
```
worst concavity              [illegible]
worst concave points         0.106094
```

```
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
```

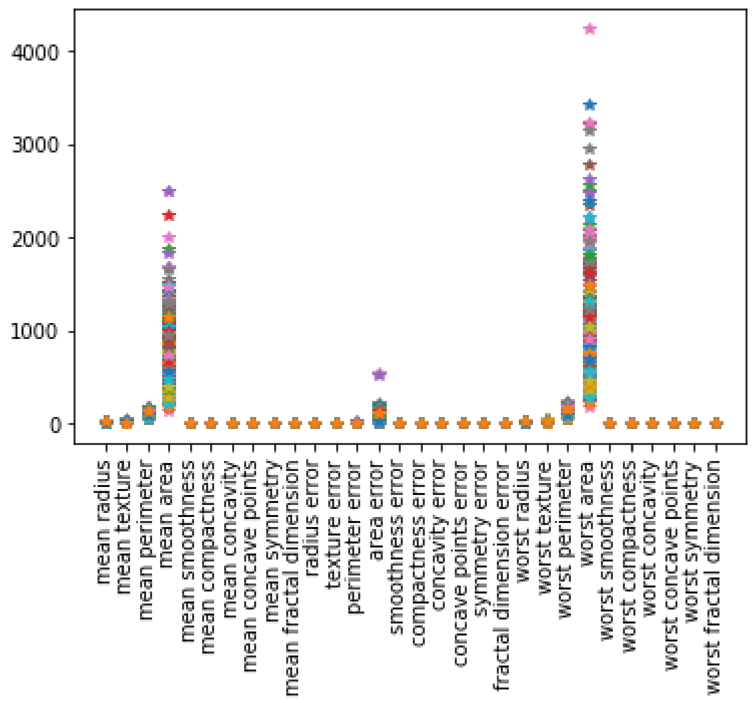
```
plt.plot(x_train,'*')
plt.show()
```



```
plt.plot(x_train.T,'*')
plt.show()
```

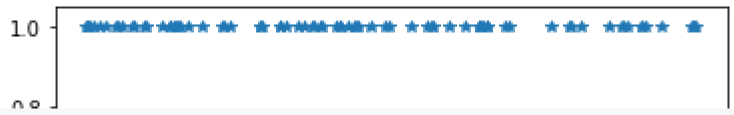


```
plt.plot(x_train.T,'*')
plt.xticks(rotation = 'vertical')
plt.show()
```



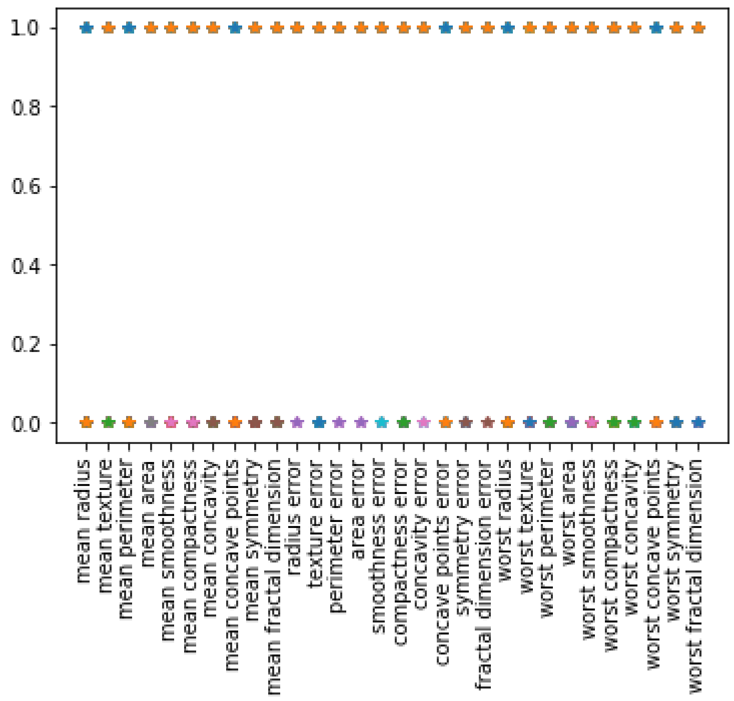
```
x_binarized_3_train = x_train['mean area'].map(lambda x:0 if x<1000 else 1)
```

```
plt.plot(x_binarized_3_train,'*')
plt.show()
```



```
x_binarized_train = x_train.apply(pd.cut,bins=2,labels=[1,0])
```

```
plt.plot(x_binarized_train.T,'*')
plt.xticks(rotation = 'vertical')
plt.show()
```



```
x_binarized_test = x_test.apply(pd.cut,bins=2,labels=[1,0])
```

```
type(x_binarized_test)
```

```
pandas.core.frame.DataFrame
```

```
x_binarized_train = x_binarized_train.values
x_binarized_test = x_binarized_test.values
```

```
type(x_binarized_test)
```

```
numpy.ndarray
```

Inference and search

```
b = 3
i = 100
if np.sum(x_binarized_train[i])>=b:
  print("inference is malignant")
else:
  print("inference is benign")
if y_train[i]==np.sum(x_binarized_train[i])>=b:
  print("ground truth is malignant")
else:
  print("ground truth is benign")
```

```
inference is malignant
ground truth is benign
```

```
from random import randint
b = 3
i = randint(0,x_binarized_train.shape[0])#100th row
if np.sum(x_binarized_train[i])>=b:
  print("inference is malignant")
else:
  print("inference is benign")
if y_train[i]==np.sum(x_binarized_train[i])>=b:
  print("ground truth is malignant")
else:
  print("ground truth is benign")
```

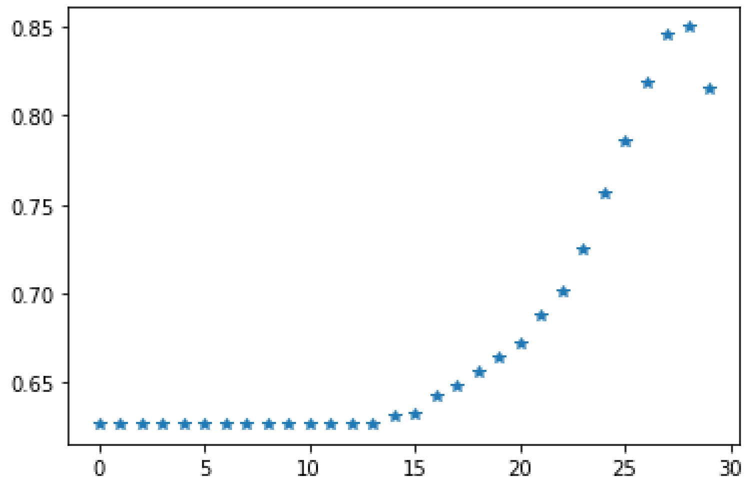
```
    inference is malignant
```

```
for b in range(0,x_binarized_train.shape[1]):
    predicted_y = []
    correct_prediction = 0
    for x,y in zip(x_binarized_train,y_train):
        pred = np.sum(x)>=b
        predicted_y.append(pred)
        if y == pred:
          correct_prediction += 1
    print(b,correct_prediction/x_binarized_train.shape[0])
```

```
0 0.626953125
1 0.626953125
2 0.626953125
3 0.626953125
4 0.626953125
5 0.626953125
6 0.626953125
7 0.626953125
8 0.626953125
9 0.626953125
10 0.626953125
11 0.626953125
12 0.626953125
13 0.626953125
14 0.630859375
15 0.6328125
16 0.642578125
17 0.6484375
18 0.65625
19 0.6640625
20 0.671875
21 0.6875
22 0.701171875
23 0.724609375
24 0.755859375
25 0.78515625
26 0.818359375
27 0.845703125
28 0.849609375
29 0.814453125
```

```
a=[]
b1=[]
for b in range(0,x_binarized_train.shape[1]):
    predicted_y = []
    correct_prediction = 0
    for x,y in zip(x_binarized_train,y_train):
        pred = np.sum(x)>=b
        predicted_y.append(pred)
        if y == pred:
          correct_prediction += 1
    a.append(b)
    b1.append(correct_prediction/x_binarized_train.shape[0])

plt.plot(a,b1,'*')
plt.show()
```



```
print(predicted_y[:10])
print(y_train[:10])
```

```
[False, True, False, True, True, False, True, False, True, False]
430    0
48     1
105    0
467    1
547    1
365    0
295    1
```

```
from sklearn.metrics import accuracy_score
b = 28
pred_values = []
for x in x_binarized_test:
  pred = np.sum(x)>=b
  pred_values.append(pred)
accuracy = accuracy_score(pred_values,y_test)
print(accuracy)
```

```
0.7894736842105263
```

```
from sklearn.metrics import accuracy_score
```

MP-Neuron Class

```
class MPNeuron:
  def __init__(self):
    self.b = None
  def model(self,x):
    return (sum(x)>=self.b)
  def predict(self,X):
    pred_values = []
    for x in X:
      pred_values.append(self.model(x))
    return np.array(pred_values)
  def fit(self,X,Y):
    accuracy = {}
    for b in range(X.shape[1]+1):
      self.b = b
      predicted_values = self.predict(X)
      accuracy[b] = accuracy_score(predicted_values,Y)
    best_b = max(accuracy,key = accuracy.get)
    self.b = best_b
    print("optimal value for b is ",best_b)
    print("optimal accuracy is ",accuracy[best_b])
```

```
mp_neuron = MPNeuron()
mp_neuron.fit(x_binarized_train,y_train)
```

```
optimal value for b is  28
optimal accuracy is  0.849609375
```

```
y_pred_test = mp_neuron.predict(x_binarized_test)
accuracy = accuracy_score(y_pred_test,y_test)
print(accuracy)
```

```
0.7894736842105263
```

```
x_train = x_train.values
x_test = x_test.values
```

Perceptron Class

```
class Perceptron:
  def __init__(self):
    self.w = None
    self.b = None
  def model(self,x):
    if np.dot(self.w,x) >= self.b:
      return 1
    else:
      return 0
  def predict(self,X):
    pred_values = []
    for x in X:
      pred = self.model(x)
      pred_values.append(pred)
    return np.array(pred_values)
  def fit(self,X,Y,epochs = 1,lr = 1):
    self.w = np.ones(X.shape[1])
    self.b = 0
    accuracy = {}
    max_accuracy = 0
    for i in range(epochs):
      for x,y in zip(X,Y):
        if y==1 and self.model(x)==0:
          self.w = self.w + lr * x
          self.b = self.b + lr * 1
        elif y==0 and self.model(x)==1:
          self.w = self.w - lr * x
          self.b = self.b - lr * 1
      accuracy[i] = accuracy_score(self.predict(X),Y)
      if(accuracy[i]>max_accuracy):
        max_accuracy = accuracy[i]
        checkpoint_w = self.w
        checkpoint_b = self.b
    self.w = checkpoint_w
    self.b = checkpoint_b
```

```
perceptron = Perceptron()
perceptron.fit(x_train,y_train,epochs=100,lr = 0.0001)
```

```
accuracy_score(perceptron.predict(x_train),y_train)
```

```
0.92578125
```

```
accuracy_score(perceptron.predict(x_test),y_test)
```

```
0.9473684210526315
```