Classification:

1. Linear regression
2. Logistic regression
3. Support vector machine

Linear regression

1. Generate 1D data synthetically
2. Take the earlier designed linear regression class
3. Find the fitting line
4. Taking 0.5 as threshold, see the classification

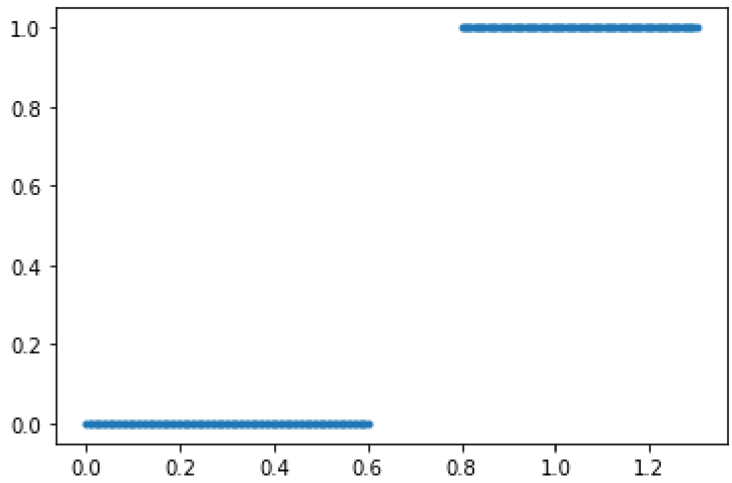
In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt

# insert your code here
```

```
(200,)
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fbb16b37198>]
```



Defining linear regression class

In [ ]:

```
# linear regression class
class lin_regression:
  # Constructor
  def __init__(self, name='reg'):
    self.name = name  # Create an instance variable

  def grad_update(self,w_old,lr,y,x):
    w=# insert your code here
    return w

  def error(self,w,y,x):
    return # insert your code here

  def mat_inv(self,y,x_aug):
    return # insert your code here
    # by Gradien descent
  def Regression_grad_des(self,x,y,lr):
    err=[]
    for i in range(1000):
     # insert your code here

    return w_pred,err
```

Data augmentation and optimal weight generation

In [ ]:

```
x=x[:,np.newaxis]
x=x.T     # to make this in M x N format, where M is the dimension
print(x.shape)
x_aug=np.concatenate((np.ones((1,x.shape[1])), x),axis=0)
print(x_aug.shape)

y=y[:,np.newaxis]

ln_reg=lin_regression()
w_opt=ln_reg.mat_inv(y,x_aug)
```

```
(1, 200)
(2, 200)
```

1. Optimal separating plane generation
2. Classification (0.5 as threshold)

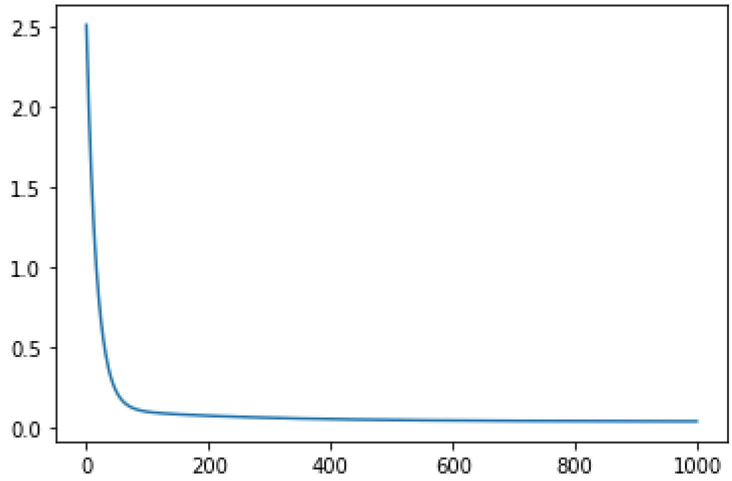
In [ ]:

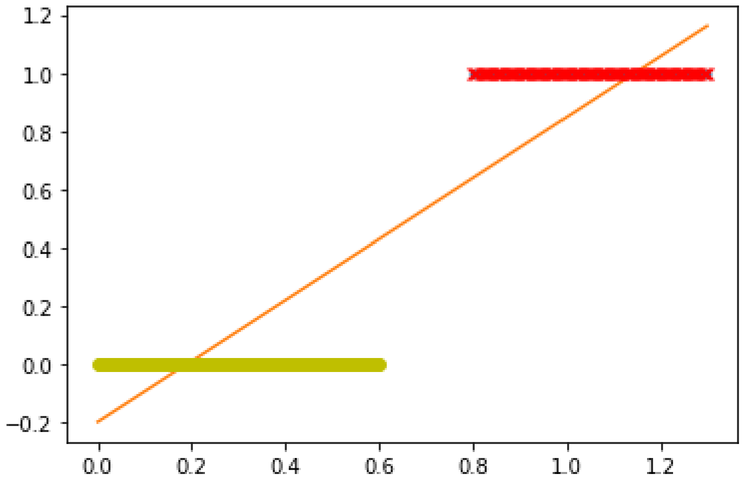
```
print(w_opt)
lr=0.01

# insert your code here
```

```
[[-0.25988351]
 [ 1.12575335]]
[[-0.20075483]
 [ 1.04780454]]
(2, 1)
(200, 1)
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fbb16819e10>]
```





Draw back of linear regression based classification

1. Generate data (have outlairs noise)
2. Find the fitting line.
3. Using 0.5 as threshold, see the classification
4. using matrix inversion (home work)

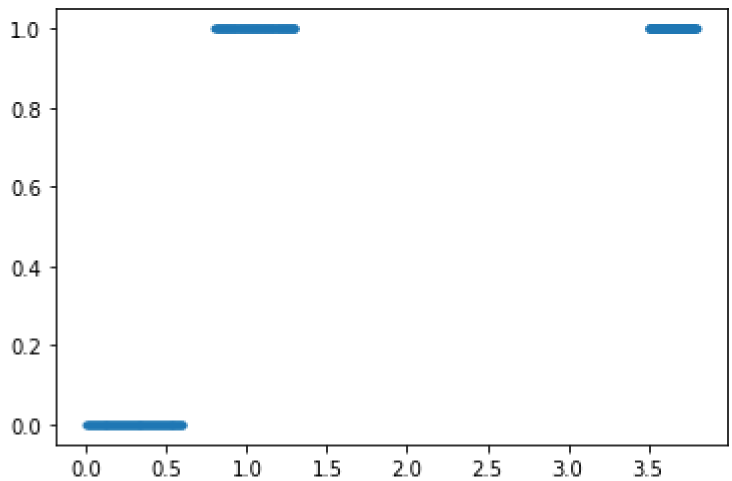
In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt

# insert your code here
```

```
(300,)
(100,)
(300,)
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fbb1672dfd0>]
```



Augment data

In [ ]:

```
# Augment data

x=x[:,np.newaxis]
y=y[:,np.newaxis]

x_aug=np.concatenate((np.ones((1,x.shape[0])), x.T),axis=0)
print(x_aug.shape)
```

```
(2, 300)
```

1. find optimal weight
2. perform classification (0.5 as threshold)

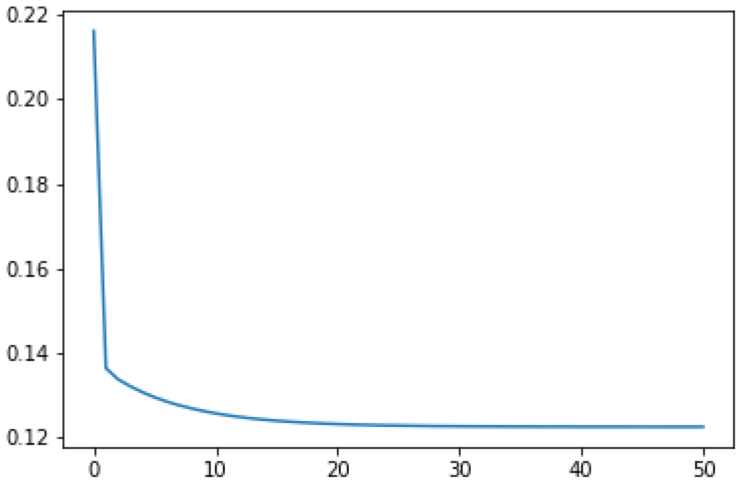
In [ ]:

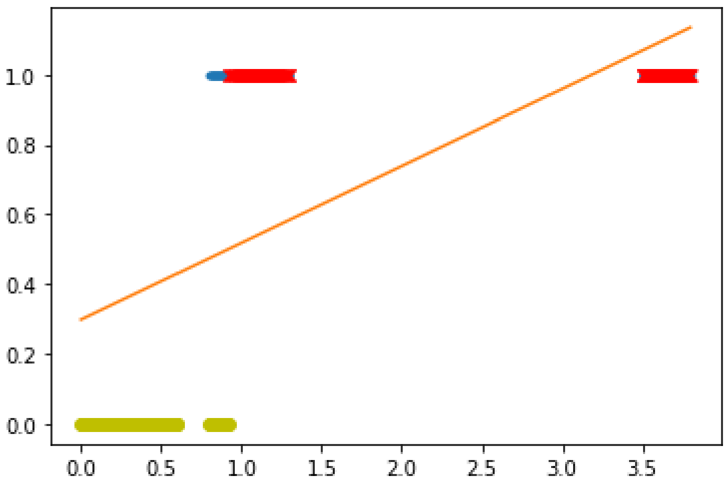
```
lr=0.1
lin_reg=lin_regression()
# insert your code here
```

```
[[0.29806582]
 [0.22033086]]
(124,)
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fbb166d6828>]
```





logistic regression

1. Error surface (logistic loss vs. MSE)
2. Solve the outlair issue
3. Circularly separable data classification
4. Multiclass classification

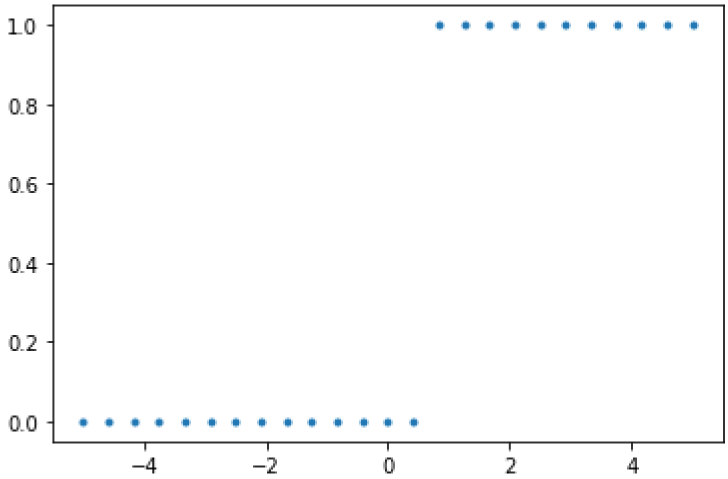
Error surface (logistic loss vs. MSE)

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt


x=np.linspace(-5,5,25)
y=np.zeros(x.shape)
y[np.where(x>0.7314)]=1

plt.plot(x,y,'.')
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fbb166646a0>]
```



1. MSE=$\frac{1}{2N}\sum_{i=1}^{N}(y_i^p - y_i)^2$, where $y^p = \frac{1}{1+e^{-w^T x}}$
2. Logistic loss=$-\frac{1}{N}\sum_{i=1}^{N} y_i log(y_i^p) + (1-y_i)log(1-y_i^p)$

In [ ]:

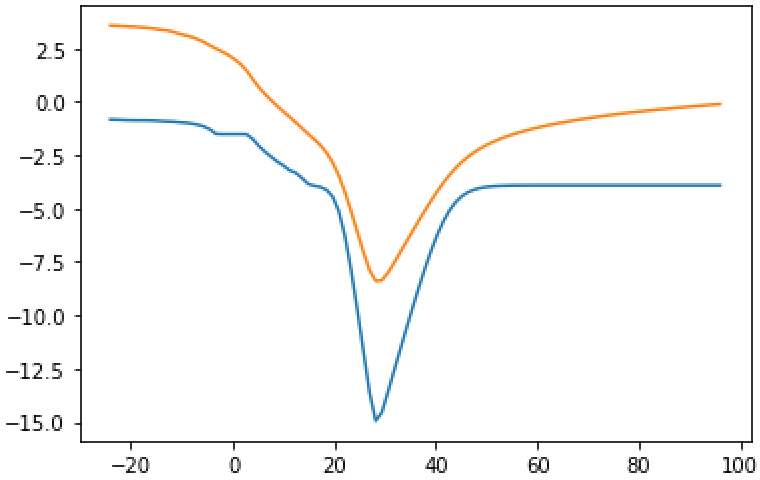
```
# search space (only w1 is searched, where as w0 is fixed)
w1_in=10/(x[1]-x[0])
w0=-w1_in*0.7314
w1=np.linspace(-w1_in,4*w1_in,100)


cost_fn_mse=[]
cost_fn_logis=[]
for i in range(w1.shape[0]):
 # insert your code here
  cost_fn_mse.append(cost_mse)
  cost_logis=# insert your code here
  cost_fn_logis.append(cost_logis)
```

In [ ]:

```
# ploting of error surface
plt.figure()
plt.plot(w1,np.log(cost_fn_mse))
plt.plot(w1,np.log(cost_fn_logis))
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fbb16601f98>]
```



Solve the outlier issue

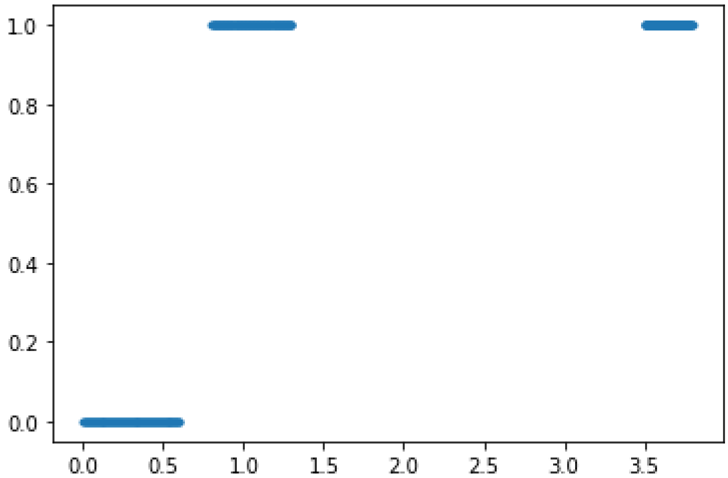
In [ ]:

```
# logistic regression
import numpy as np
import matplotlib.pyplot as plt

# insert your code here

plt.figure()
plt.plot(x,y,'.')
```

```
(300,)
(100,)
(300,)
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fbb165326a0>]
```



In [ ]:

```
class logis_regression:
  # Constructor
  def __init__(self, name='reg'):
    self.name = name  # Create an instance variable

  def logis(self,x,w_old):
    # insert your code here
    return op

  def grad_update(self,w_old,lr,y,x):
    # insert your code here
    return w

  def error(self,w,y,x):
    return # insert your code here

    # by Gradien descent
  def Regression_grad_des(self,x,y,lr):
    err=[]
    for i in range(1000):
      # insert your code here
      if dev<=10**(-20):
        break

    return w_pred,err
```

In [ ]:

```
# augmentation and data formating

x=x[:,np.newaxis]
y=y[:,np.newaxis]
print(x.shape)
x_aug=np.concatenate((np.ones((1,x.shape[0])), x.T),axis=0)
```

```
(300, 1)
```

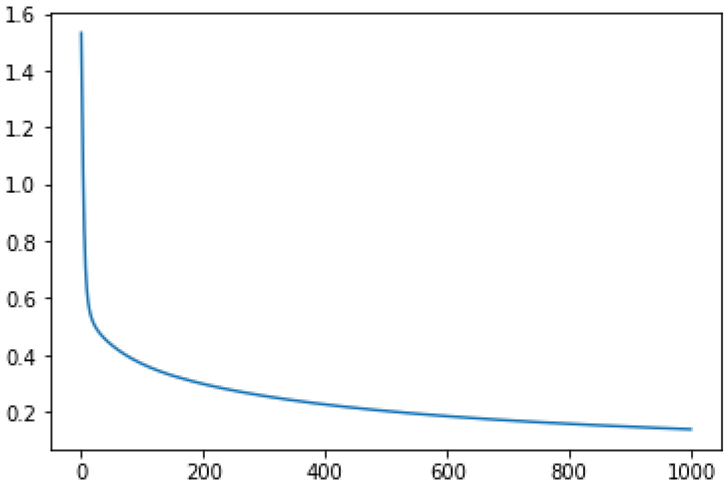
In [ ]:

```
log_reg=logis_regression()
w_pred,err=log_reg.Regression_grad_des(x_aug,y,0.1)
print(w_pred)

plt.plot(err)
```

```
[[-2.86836608]
 [ 4.47612186]]
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fbb165227b8>]
```



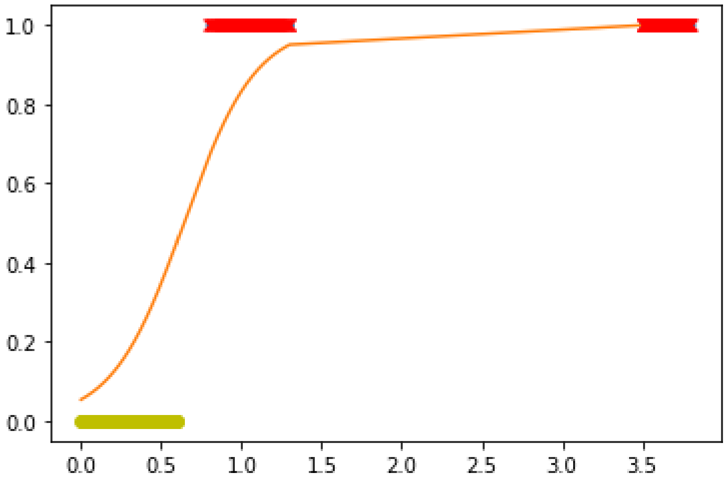
In [ ]:

```
# output computation
# insert your code here

plt.plot(x0,np.zeros((x0).shape),'o',color='y')
plt.plot(x1,np.ones((x1).shape),'x',color='r')
```

```
(100,)
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fbb164bf2b0>]
```



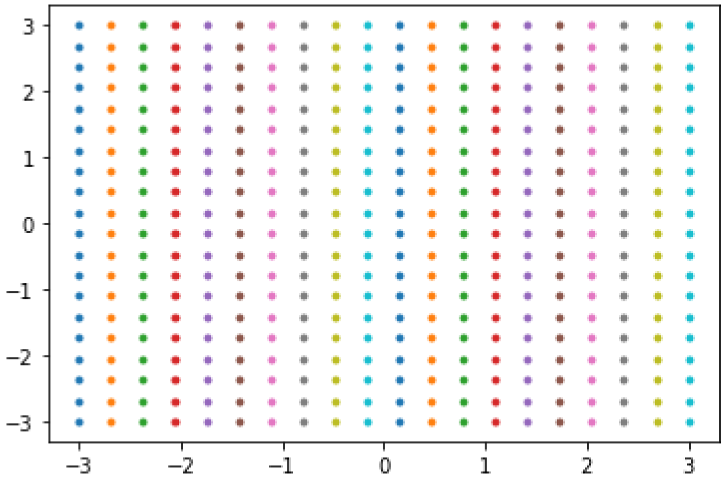
Classification of circularly separated data using logistic regression

In [ ]:

```
# Generating circularly separated data
import numpy as np
import matplotlib.pyplot as plt


x1=np.linspace(-3,3,20)
x2=np.linspace(-3,3,20)

x11,x22=np.meshgrid(x1,x2)
plt.plot(x11,x22,'.')
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fbb163f4828>,
 <matplotlib.lines.Line2D at 0x7fbb163f4908>,
 <matplotlib.lines.Line2D at 0x7fbb163f4a58>,
 <matplotlib.lines.Line2D at 0x7fbb163f4ba8>,
 <matplotlib.lines.Line2D at 0x7fbb163f4cf8>,
 <matplotlib.lines.Line2D at 0x7fbb163f4e48>,
 <matplotlib.lines.Line2D at 0x7fbb163f4f98>,
 <matplotlib.lines.Line2D at 0x7fbb163ff128>,
 <matplotlib.lines.Line2D at 0x7fbb163ff278>,
 <matplotlib.lines.Line2D at 0x7fbb163ff3c8>,
 <matplotlib.lines.Line2D at 0x7fbb164514e0>,
 <matplotlib.lines.Line2D at 0x7fbb163ff630>,
 <matplotlib.lines.Line2D at 0x7fbb163ff780>,
 <matplotlib.lines.Line2D at 0x7fbb163ff8d0>,
 <matplotlib.lines.Line2D at 0x7fbb163ffa20>,
 <matplotlib.lines.Line2D at 0x7fbb163ffb70>,
 <matplotlib.lines.Line2D at 0x7fbb163ffcc0>,
 <matplotlib.lines.Line2D at 0x7fbb163ffe10>,
 <matplotlib.lines.Line2D at 0x7fbb163fff60>,
 <matplotlib.lines.Line2D at 0x7fbb164040f0>]
```



1. Circularly separated data generation

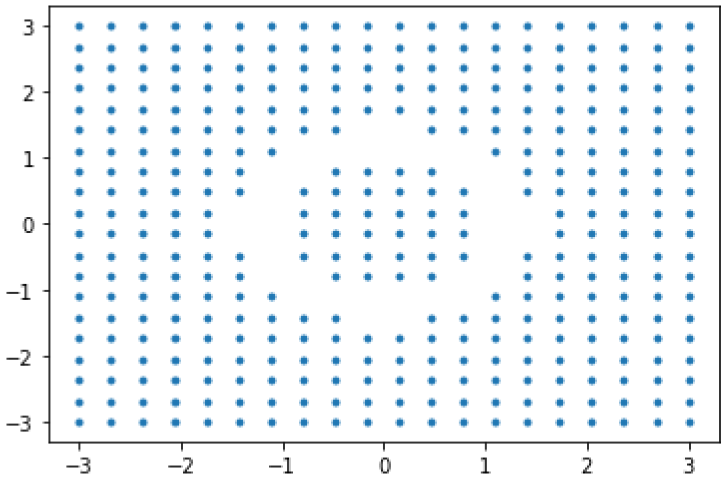
In [ ]:

```
x1=x11.flatten()
x2=x22.flatten()

x=np.concatenate((x1[:,np.newaxis],x2[:,np.newaxis]),axis=1) # to make matrix fo
rmat
print(x.shape)

aind=np.where((x[:,0]**(2)+x[:,1]**(2))<=0.9)

bind=np.where((x[:,0]**(2)+x[:,1]**(2))>=2.2)

x1=x[aind[0],:]
x2=x[bind[0],:]
print(x1.shape)
print(x2.shape)
x=np.concatenate((x1,x2))
print(x.shape)

plt.plot(x[:,0],x[:,1],'.')
```

```
(400, 2)
(32, 2)
(332, 2)
(364, 2)
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fbb16393d68>]
```



As in case of circularly separated data, the boundary is nonlinear, so squred feature is taken.

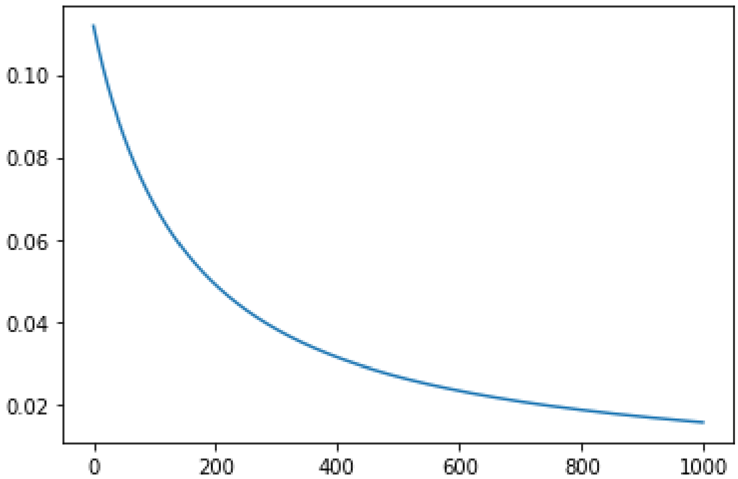
In [ ]:

```
# perform logistic regression

y1=np.zeros((x1.shape[0]))
y2=np.ones((x2.shape[0]))
y=np.concatenate((y1,y2))

x_aug=# insert your code here    # squring to learn circular separation
print(x_aug.shape)

log_reg=logis_regression()
w_pred,err=log_reg.Regression_grad_des(x_aug,y[:,np.newaxis],0.3)
plt.plot(err)
```

```
(3, 364)
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fbb1669c320>]
```



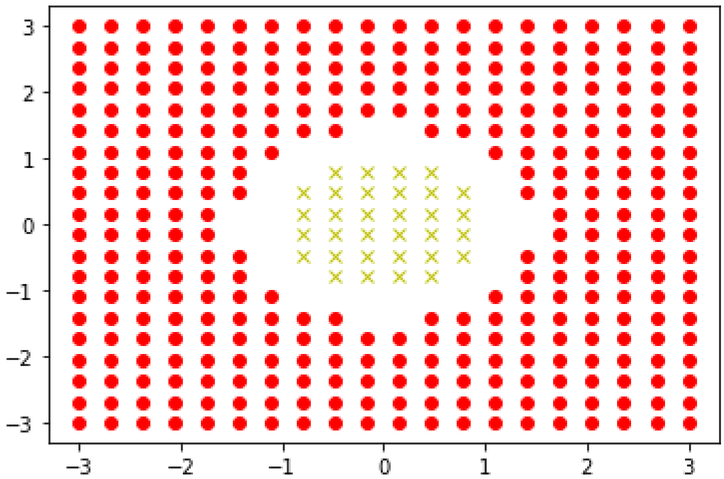
Plot classification using 0.5 as threshold

In [ ]:

```
y_pred=log_reg.logis(x_aug,w_pred)

# insert your code here

x00=x[ind1,:]
x11=x[ind2,:]

plt.figure()
plt.plot(x00[:,0],x00[:,1],'x',color='y')
plt.plot(x11[:,0],x11[:,1],'o',color='r')
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fbb164ad358>]
```



Multiclass logistic regression

1. Generate 1D data with 3 classes

One vs rest classification

1. lets take polynomial of order 2 (by seeing the data distribution)

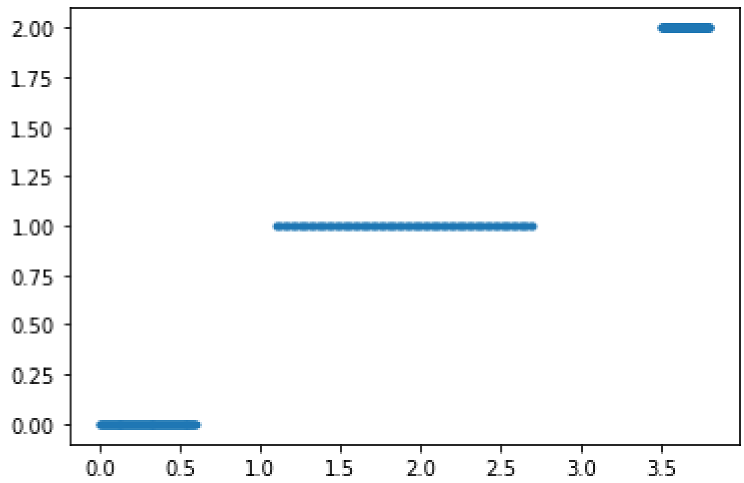
In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt

x1=np.linspace(0,0.6,100)
x2=np.linspace(1.1,2.7,100)
x3=np.linspace(3.5,3.8,100)

x=np.concatenate((x1,x2,x3))
print(x.shape)

y1=np.zeros(x1.shape)
y2=np.ones(x2.shape)
y3=np.tile([2],x3.shape)

y=np.concatenate((y1,y2,y3))

plt.figure()
plt.plot(x,y,'.')
```

```
(300,)
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fbb167d7470>]
```



In [ ]:

```
def data_transform(X,degree):
  X_new=[]
  for i in range(degree +1):
    X_new.append(X**i)
  X_new = np.concatenate(X_new)
  return X_new
```

In [ ]:

```
x_aug=data_transform(x[np.newaxis,:],2)
print(x_aug.shape)
```

```
(3, 300)
```

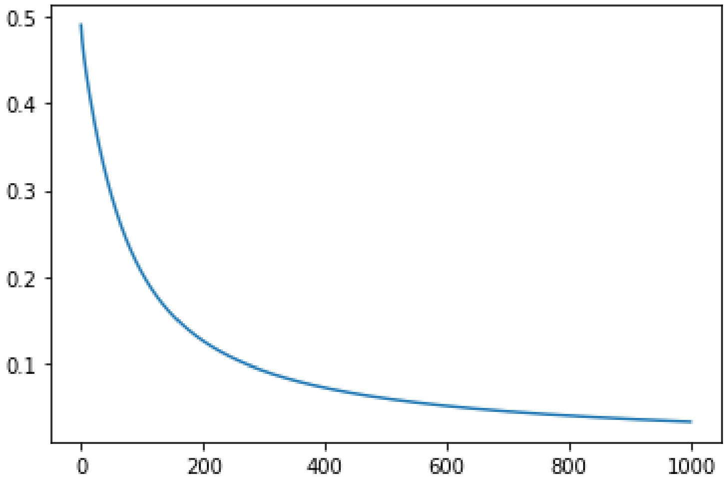
In [ ]:

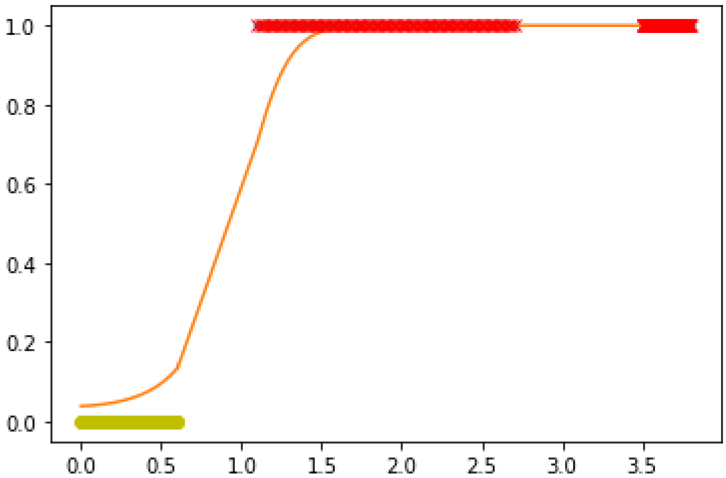
```
# plot for classification
def plot_op(x,y_pred):
  # insert your code here

  x0=x[ind0,:]
  x1=x[ind1,:]


  plt.plot(x0,np.zeros((x0).shape),'o',color='y')
  plt.plot(x1,np.ones((x1).shape),'x',color='r')
```

In [ ]:

```
# take class 0 as '0' and other to '1'
# insert your code here


plt.plot(err)
print(w1_pred)


# ploting
plt.figure()
plt.plot(x,y1_mod,'.')

y1_pred=log_reg.logis(x_aug,w1_pred)
plt.plot(x,y1_pred[:,0])
plot_op(x[:,np.newaxis],y1_pred)
```

```
[[-3.21347587]
 [ 0.46352272]
 [ 2.96765079]]
```

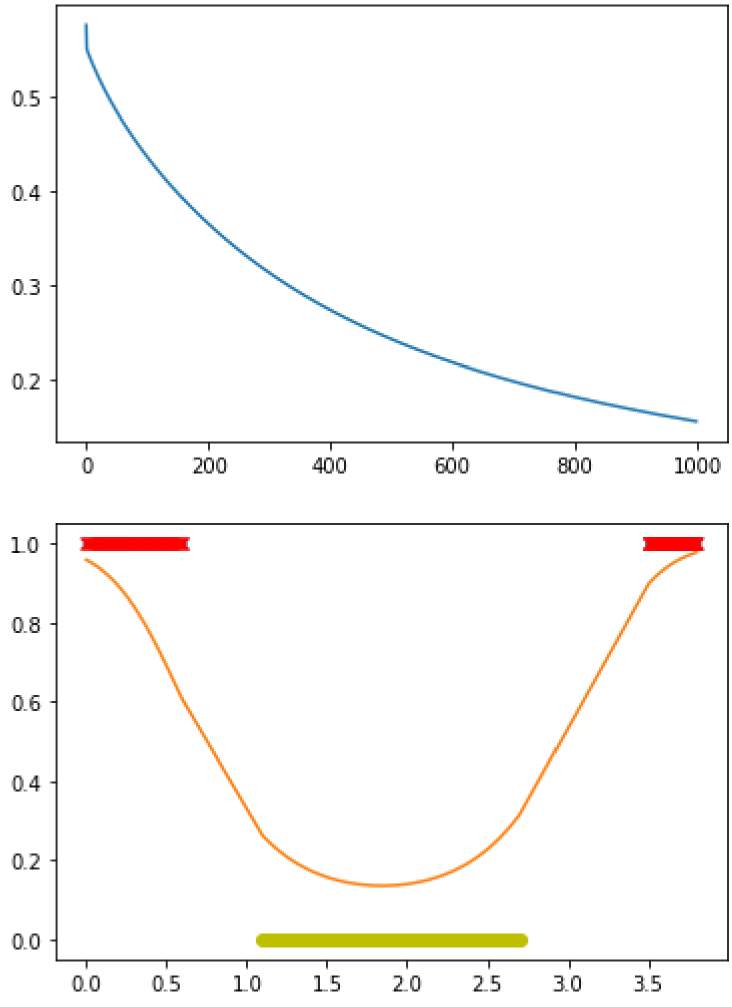




In [ ]:

```
# take class 1 as '0' and other to '1'
# insert your code here



# ploting
plt.figure()
plt.plot(x,y2_mod,'.')

y2_pred=log_reg.logis(x_aug,w2_pred)
plt.plot(x,y2_pred[:,0])
plot_op(x[:,np.newaxis],y2_pred)
```

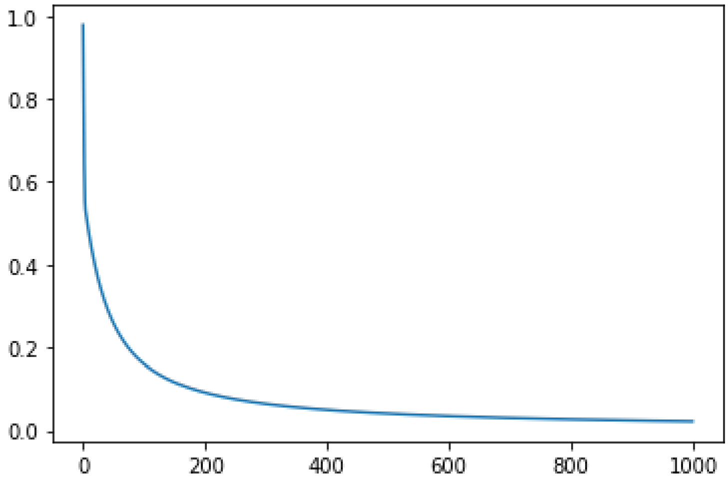
```
[[ 3.14888913]
 [-5.4202842 ]
 [ 1.46856098]]
```

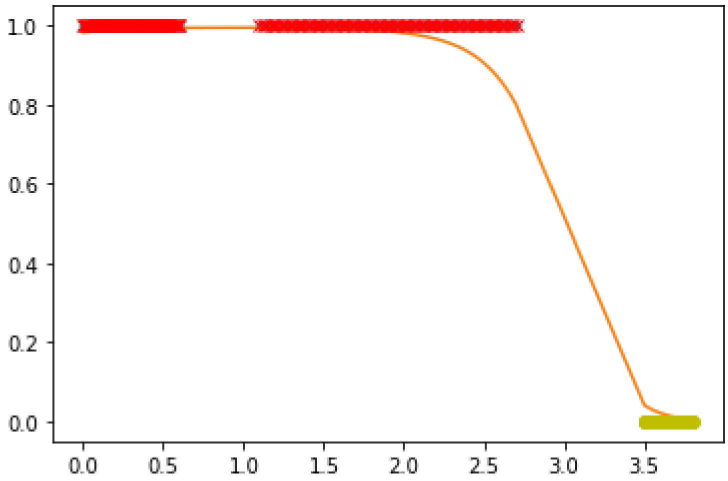


In [ ]:

```
# take class 2 as '0' and other to '1'
# insert your code here
# ploting
plt.figure()
plt.plot(x,y3_mod,'.')


y3_pred=log_reg.logis(x_aug,w3_pred)
plt.plot(x,y3_pred[:,0])

plot_op(x[:,np.newaxis],y3_pred)
```

```
[[ 4.02245634]
 [ 2.66007096]
 [-1.34592077]]
```



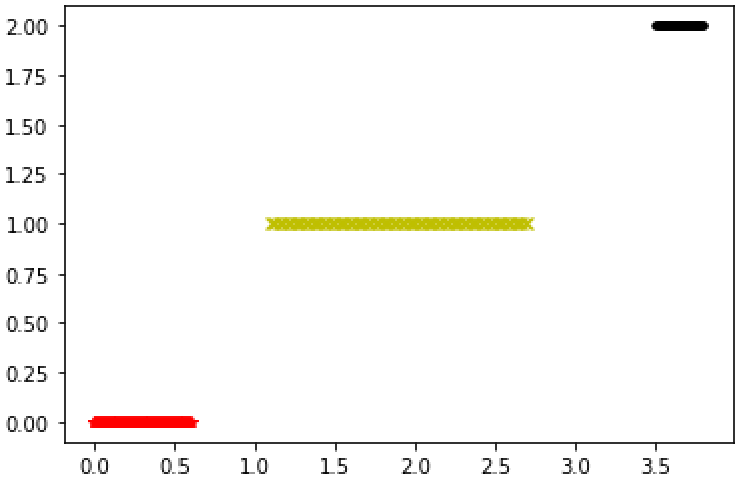


In [ ]:

```
# final classification

# insert your code here  # as '0' is taken as referance
# insert your code here

plt.figure()
plt.plot(x1,np.zeros(x1.shape),'*',color='r')
plt.plot(x2,np.ones(x2.shape),'x',color='y')
plt.plot(x3,np.tile([2],x3.shape),'.',color='k')
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fbb16168cc0>]
```



Support vector machine

1. Try to maximize the margin of separation between data.
2. Instead of learning wx+b=0 separating hyperplane directly (like logistic regression), SVM try to learn wx+b=0, such that, the margin between two hyperplanes wx+b=1 and wx+b=-1 (also known as support vectors) is maximum.
3. Margin between wx+b=1 and wx+b=-1 hyperplane is $\frac{2}{||w||}$
4. we have a constraint optimization problem of maximizing $\frac{2}{||w||}$, with constraints wx+b>=1 (for +ve class) and wx+b<=-1 (for -ve class).
5. As $y_i = 1$ for +ve class and $y_i = -1$ for -ve class, the constraint can be re-written as:
$$y(wx + b) >= 1$$
6. Final optimization is (i.e to find w and b):
$$\min_{||w||} \frac{1}{2}||w||,$$
$$y(wx + b) \geq 1, \; \forall \; data$$

Acknowledgement:

https://pythonprogramming.net/predictions-svm-machine-learning-tutorial/ (https://pythonprogramming.net/predictions-svm-machine-learning-tutorial/)

https://medium.com/deep-math-machine-learning-ai/chapter-3-1-svm-from-scratch-in-python-86f93f853dc (https://medium.com/deep-math-machine-learning-ai/chapter-3-1-svm-from-scratch-in-python-86f93f853dc)

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import math
```

Data generation:

1. Generate 2D gaussian data with fixed mean and variance for 2 class.(var=Identity, class1: mean[-4,-4], class2: mean[1,1], No. of data 25 from each class)
2. create the label matrix
3. Plot the generated data

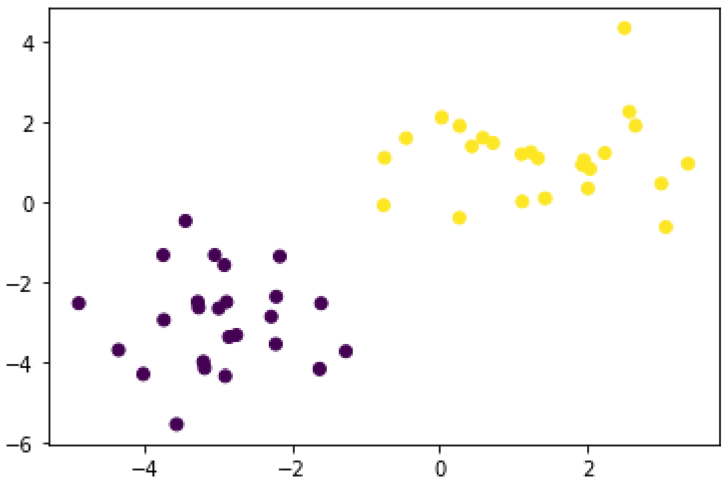
In [ ]:

```
No_sample=50
mean1=np.array([-3,-3])
var1=np.array([[1,0],[0,1]])
mean2=np.array([1,1])
var2=var1
data1=np.random.multivariate_normal(mean1,var1,int(No_sample/2))
data2=np.random.multivariate_normal(mean2,var2,int(No_sample/2))
X=np.concatenate((data1,data2))
print(X.shape)
y=np.concatenate((-1*np.ones(data1.shape[0]),np.ones(data2.shape[0])))
print(y.shape)

plt.figure()
plt.scatter(X[:,0],X[:,1],marker='o',c=y)
```

```
(50, 2)
(50,)
```

Out[ ]:

```
<matplotlib.collections.PathCollection at 0x7feb64e2e4e0>
```



Create a data dictionary, which contains both label and data points.

In [ ]:

```
postiveX=[]
negativeX=[]
for i,v in enumerate(y):
    if v==-1:
        negativeX.append(X[i])
    else:
        postiveX.append(X[i])

#our data dictionary
data_dict = {-1:np.array(negativeX), 1:np.array(postiveX)}
```

SVM training

1. create a search space for w (i.e w1=w2),[0, 0.5*max((abs(feat)))] and for b, [-max((abs(feat))),max((abs(feat)))], with appropriate step.
2. we will start with a higher step and find optimal w and b, then we will reduce the step and again re-evaluate the optimal one.
3. In each step, we will take transform of w, [1,1], [-1,1],[1,-1] and [-1,-1] to search arround the w.
4. In every pass (for a fixed step size) we will store all the w, b and its corresponding ||w||, which make the data correctly classified as per the condition $y(wx+b) \geq 1$.
5. Obtain the optimal hyperplane having minimum ||w||.
6. Start with the optimal w and repeat the same (step 3,4 and 5) for a reduced step size.

In [ ]:

```
# it is just a searching algorithem, not a complicated optimization algorithem,
 (just for understanding of concepts through visualization)

def SVM_Training(data_dict):

    # insert your code here

    return w,b
```

Training

In [ ]:

```
#all the required variables
w=[] #weights 2 dimensional vector
b=[] #bias
w,b=SVM_Training(data_dict)
print(w)
print(b)
```

```
[0.75384537 0.75384537]
1.6351792900618864
```

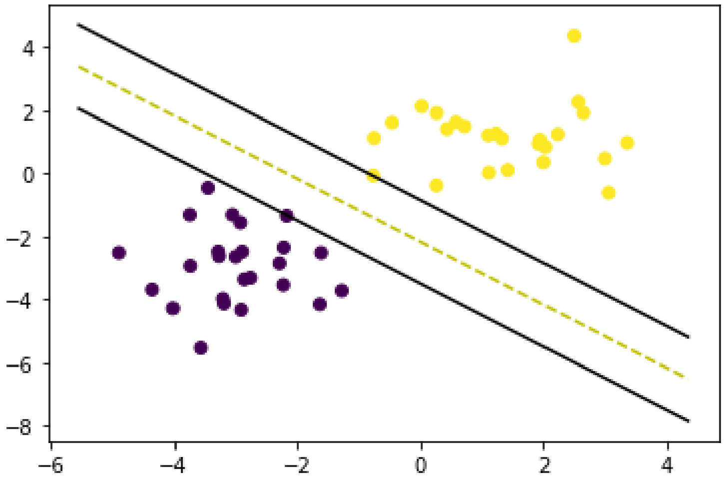
Visualization of the SVM separating hyperplanes (after training)

In [ ]:

```
def visualize(data_dict):




        plt.scatter(X[:,0],X[:,1],marker='o',c=y)

        # hyperplane = x.w+b
        # v = x.w+b
        # psv = 1
        # nsv = -1
        # dec = 0
        def hyperplane_value(x,w,b,v):
            return (-w[0]*x-b+v) / w[1]


        hyp_x_min = np.min([np.min(data_dict[1]),np.min(data_dict[-1])])
        hyp_x_max = np.max([np.max(data_dict[1]),np.max(data_dict[-1])])

        # (w.x+b) = 1
        # positive support vector hyperplane
        psv1 = hyperplane_value(hyp_x_min, w, b, 1)
        psv2 = hyperplane_value(hyp_x_max, w, b, 1)
        plt.plot([hyp_x_min,hyp_x_max],[psv1,psv2], 'k')

        # (w.x+b) = -1
        # negative support vector hyperplane
        nsv1 = hyperplane_value(hyp_x_min, w, b, -1)
        nsv2 = hyperplane_value(hyp_x_max, w, b, -1)
        plt.plot([hyp_x_min,hyp_x_max],[nsv1,nsv2], 'k')

        # (w.x+b) = 0
        # positive support vector hyperplane
        db1 = hyperplane_value(hyp_x_min, w, b, 0)
        db2 = hyperplane_value(hyp_x_max, w, b, 0)
        plt.plot([hyp_x_min,hyp_x_max],[db1,db2], 'y--')
```

In [ ]:

```
fig = plt.figure()
visualize(data_dict)
```



Testing

1. Generate test data as like training
2. See the classification
3. if $wx_{test} + b > 0$, $y_{test} = 1$ else $y_{test} = -1$

In [ ]:

```
def predict(data,w,b):
  y_pred=np.sign(np.dot(data,w)+b)
  return y_pred
```

In [ ]:

```
No_test_sample=40
data1=np.random.multivariate_normal(mean1,var1,int(No_test_sample/2))
data2=np.random.multivariate_normal(mean2,var2,int(No_test_sample/2))
test_data=np.concatenate((data1,data2))
y_gr=np.concatenate((-1*np.ones(data1.shape[0]),np.ones(data2.shape[0])))

# evaluate with the trained model

y_pred=predict(test_data,w,b)
accuracy=# insert your code here
print('test accuracy=',accuracy)

#  Visualization
plt.figure()
visualize(data_dict)
plt.scatter(test_data[:,0],test_data[:,1],marker='x',c=y_gr)
```

```
test accuracy= 100.0
```

Out[ ]:

```
<matplotlib.collections.PathCollection at 0x7feb64ddd748>
```