Load Data

```
import pandas as pn
url ='https://raw.githubusercontent.com/swakkhar/'
filename='MachineLearning/master/Codes/logistic.csv'
data = pn.read_csv(url+filename,header=None)
```

```
import numpy as np
data=np.asarray(data)
X = np.delete(data, data.shape[1] - 1, axis=1)
y = data[:, -1]
print("X.shape:",X.shape)
print("y.shape:",y.shape)
```

```
X.shape: (100, 2)
y.shape: (100,)
```

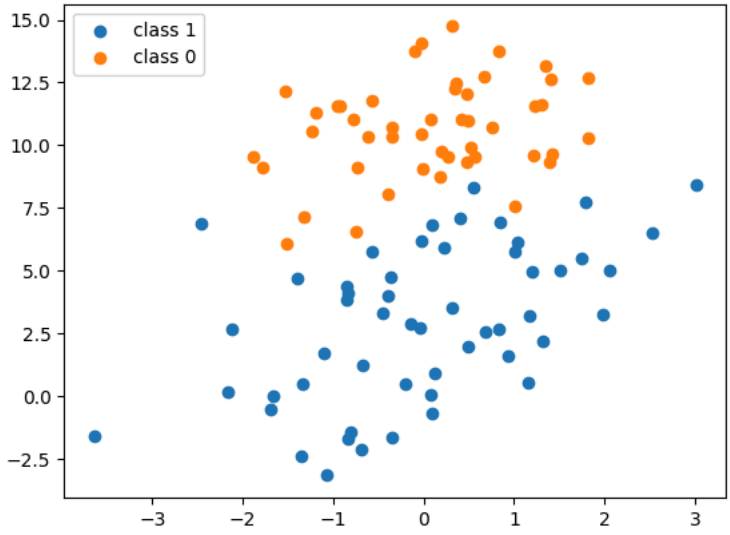
```
for i in range(X.shape[0]):
  print(X[i,0],",",X[i,1],",",y[i])
```

```
-0.017612 , 14.053064 , 0.0
-1.395634 , 4.662541 , 1.0
-0.752157 , 6.53862 , 0.0
-1.322371 , 7.152853 , 0.0
0.423363 , 11.054677 , 0.0
0.406704 , 7.067335 , 1.0
0.667394 , 12.741452 , 0.0
-2.46015 , 6.866805 , 1.0
0.569411 , 9.548755 , 0.0
-0.026632 , 10.427743 , 0.0
0.850433 , 6.920334 , 1.0
1.347183 , 13.1755 , 0.0
1.176813 , 3.16702 , 1.0
-1.781871 , 9.097953 , 0.0
-0.566606 , 5.749003 , 1.0
0.931635 , 1.589505 , 1.0
-0.024205 , 6.151823 , 1.0
-0.036453 , 2.690988 , 1.0
-0.196949 , 0.444165 , 1.0
1.014459 , 5.754399 , 1.0
1.985298 , 3.230619 , 1.0
-1.693453 , -0.55754 , 1.0
-0.576525 , 11.778922 , 0.0
-0.346811 , -1.67873 , 1.0
-2.124484 , 2.672471 , 1.0
1.217916 , 9.597015 , 0.0
-0.733928 , 9.098687 , 0.0
-3.642001 , -1.618087 , 1.0
0.315985 , 3.523953 , 1.0
1.416614 , 9.619232 , 0.0
-0.386323 , 3.989286 , 1.0
0.556921 , 8.294984 , 1.0
1.224863 , 11.58736 , 0.0
-1.347803 , -2.406051 , 1.0
1.196604 , 4.951851 , 1.0
0.275221 , 9.543647 , 0.0
0.470575 , 9.332488 , 0.0
-1.889567 , 9.542662 , 0.0
-1.527893 , 12.150579 , 0.0
-1.185247 , 11.309318 , 0.0
-0.445678 , 3.297303 , 1.0
1.042222 , 6.105155 , 1.0
-0.618787 , 10.320986 , 0.0
1.152083 , 0.548467 , 1.0
0.828534 , 2.676045 , 1.0
-1.237728 , 10.549033 , 0.0
-0.683565 , -2.166125 , 1.0
0.229456 , 5.921938 , 1.0
-0.959885 , 11.555336 , 0.0
0.492911 , 10.993324 , 0.0
0.184992 , 8.721488 , 0.0
-0.355715 , 10.325976 , 0.0
-0.397822 , 8.058397 , 0.0
0.824839 , 13.730343 , 0.0
1.507278 , 5.027866 , 1.0
0.099671 , 6.835839 , 1.0
-0.344008 , 10.717485 , 0.0
1.785928 , 7.718645 , 1.0
```

Visualize data

```
import matplotlib
import matplotlib.pyplot as plt

classA=[]
classB=[]
for i in range(len(y)):
    if y[i]==1.0:
        classA.append(X[i,:])
    else:
        classB.append(X[i, :])
a=plt.scatter(np.asarray(classA)[:,0],np.asarray(classA)[:,-1])

b=plt.scatter(np.asarray(classB)[:,0],np.asarray(classB)[:,-1])
plt.legend((a,b),('class 1','class 0'),loc='upper left')
```

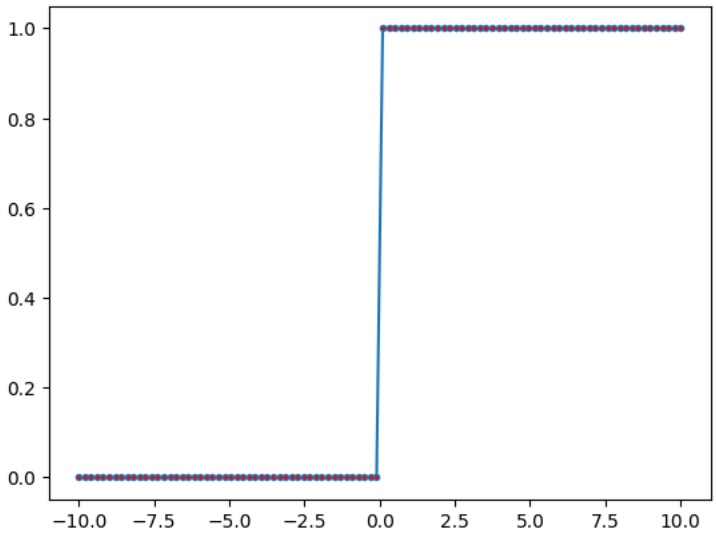
```
<matplotlib.legend.Legend at 0x78b469f03fd0>
```



Closer look at the step function

```
def stepLR(x):
  if x > 0:
    return 1
  else:
    return 0

somex=np.linspace(-10,10,100)
somey=[stepLR(e) for e in somex]
import matplotlib
import matplotlib.pyplot as plt

plt.plot(somex,somey,marker=".",markerfacecolor='r')
```

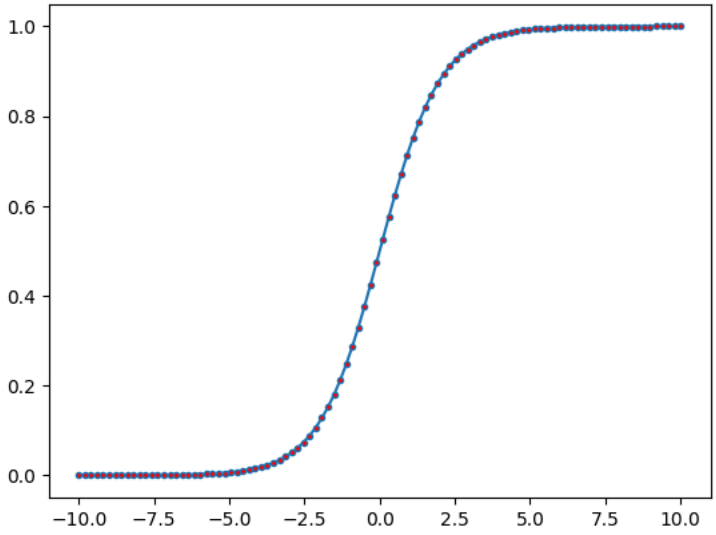
```
[<matplotlib.lines.Line2D at 0x78b469c20cd0>]
```



Sigmoid Function

```
import math
def sigmoid(x):
  return 1/(1+math.exp(-x))
somex=np.linspace(-10,10,100)
somey=[sigmoid(e) for e in somex]
import matplotlib
import matplotlib.pyplot as plt

plt.plot(somex,somey,marker=".",markerfacecolor='r')
```

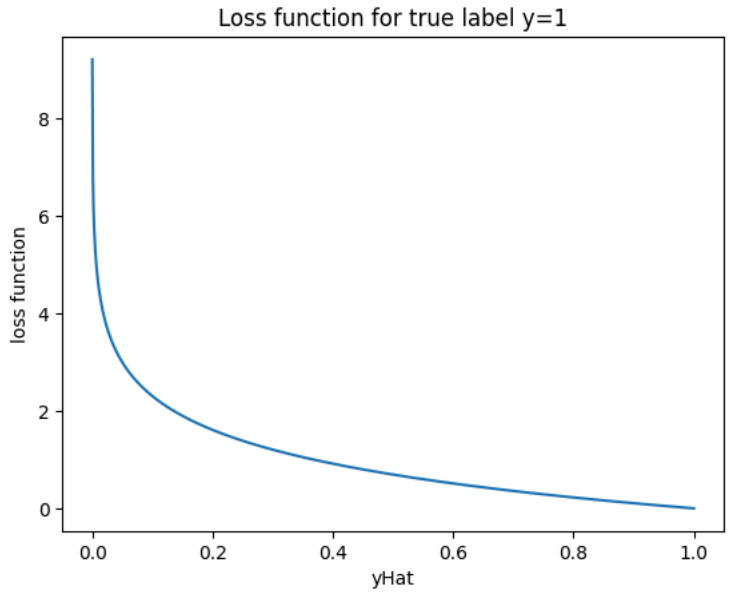
```
[<matplotlib.lines.Line2D at 0x78b467c43160>]
```

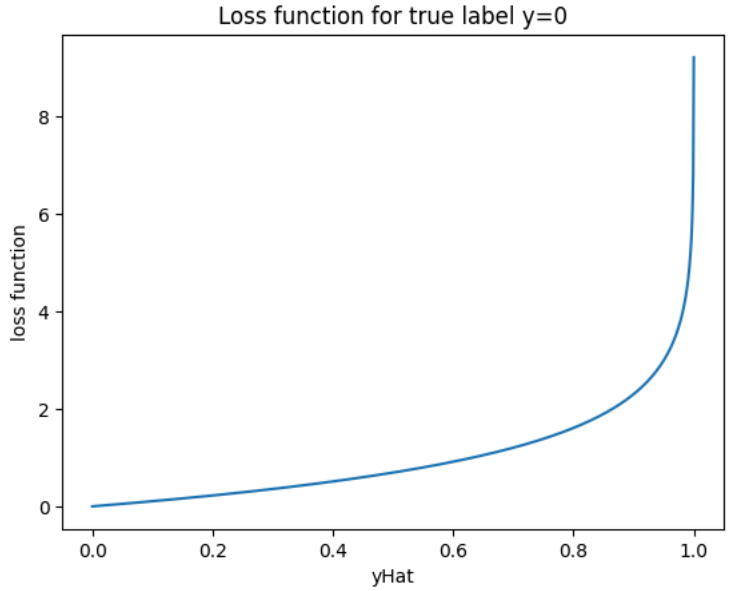


Cross Entropy

```
def crossEntropy(y,yHat):
  if y == 1:
    return -math.log(yHat)
  else:
    return -math.log(1 - yHat)
```

```
yhat=np.linspace(0.0001,.9999,1000)
loss = [crossEntropy(1,r) for r in yhat]
plt.plot(yhat,loss)
plt.xlabel("yHat")
plt.ylabel("loss function")
plt.title("Loss function for true label y=1")
plt.show()


yhat=np.linspace(0.0001,.9999,1000)
loss = [crossEntropy(0,r) for r in yhat]
plt.plot(yhat,loss)
plt.xlabel("yHat")
plt.ylabel("loss function")
plt.title("Loss function for true label y=0")
plt.show()
```



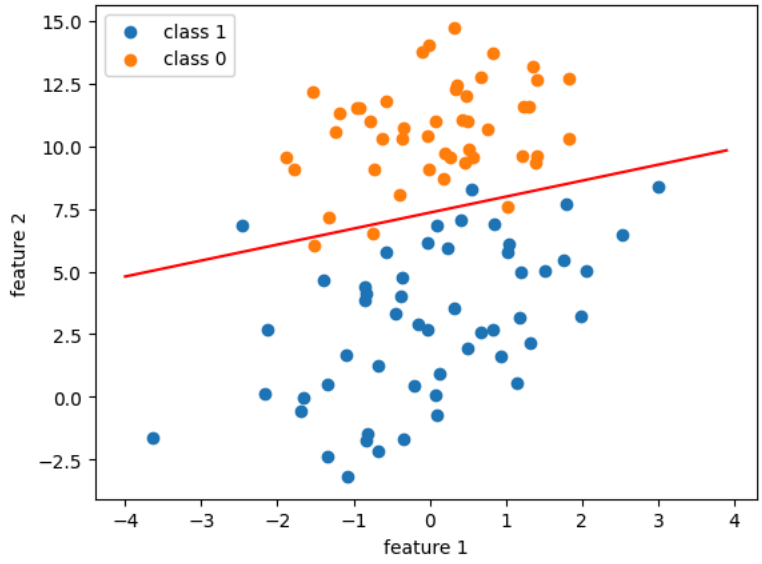


Same gradient descent works!

```
def sigmoid(z):
    return 1.0/(1+np.exp(-z))


import copy
def learnWeights(X,y,maxIter,alpha):
  ones = np.ones((X.shape[0],1))
  # Deep copy the parameters
  X=copy.deepcopy(X)
  y=copy.deepcopy(y)
  X=np.concatenate((ones,X),axis=1)
  X=np.mat(X)
  y=np.mat(y)
  w=np.random.rand(X.shape[1],1)
  #w = np.ones((X.shape[1], 1))
  for i in range(0,maxIter):
    # predict y
    z=X*w
    predy=sigmoid(z)
    delY = predy-y.T
    delw = X.T * delY
    w = w - delw * alpha
  return w

print(X.shape)
print(y.shape)
w=learnWeights(X,y,1000,0.01)
print(X.shape)
```

```
print(y.shape)

print(w)

classA=[]
classB=[]
i=0
for index in y:
    if index==1.0:
        classA.append(X[i,:])
    else:
        classB.append(X[i, :])
    i+=1
a=plt.scatter(np.asarray(classA)[:,0],np.asarray(classA)[:,-1])

b=plt.scatter(np.asarray(classB)[:,0],np.asarray(classB)[:,-1])
plt.legend((a,b),('class 1','class 0'),loc='upper left')


testx = np.arange(-4.0, 4.0, 0.1)

testy = (-w[0]-w[1]*testx)/w[2]
plt.plot(testx,testy.T,color="r")

plt.xlabel("feature 1")
plt.ylabel("feature 2")
plt.show()
```

```
(100, 2)
(100,)
(100, 2)
(100,)
[[13.2963642 ]
 [ 1.15381353]
 [-1.80876868]]
```




```
from sklearn.linear_model import LogisticRegression
```

```
clf = LogisticRegression(random_state=0).fit(X, y)
```

```
clf.predict([[1,1]])
```

```
array([1.])
```

```
clf.predict_proba([[1,15]])
```

```
array([[9.99981333e-01, 1.86667611e-05]])
```

```
clf.score(X, y)
```

```
0.95
```

A lighter algorithm for learning weights

```
# This is formatted as code
```

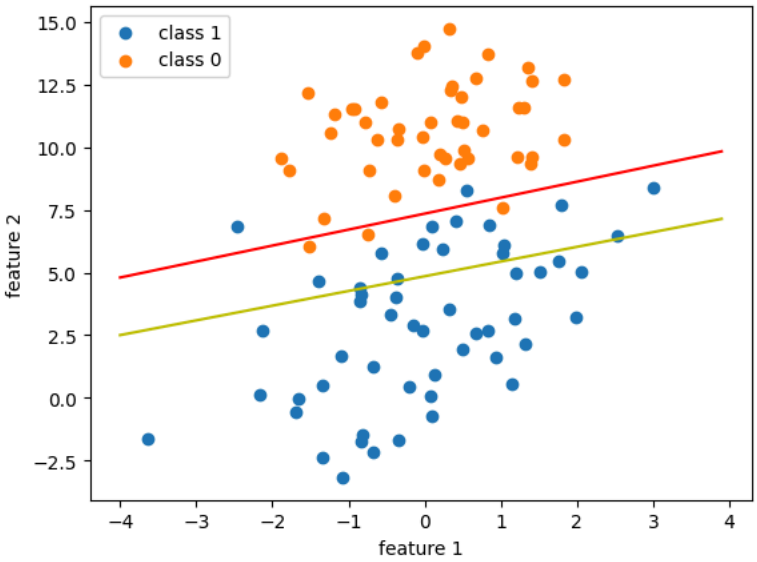
```
import random
def NaiveStocGradDescent(X, y, numIter=150):
  X=copy.deepcopy(X)
  y=copy.deepcopy(y)
  m,n = X.shape
  ones = np.ones((X.shape[0], 1))
  X = np.concatenate((ones, X), axis=1)
  X = np.mat(X)
  y = np.mat(y)
  w = np.random.rand(X.shape[1], 1)
  y=y.T

  alpha = 0.01

  for j in range(10):

    for i in range(m):
      i = int(random.uniform(0,m))
      h = sigmoid((X[i])*w)
      error = y[i] - h
      w = w + alpha * (error * X[i]).T
  return w

print(X.shape)
print(y.shape)
w1=NaiveStocGradDescent(X,y)
print(w1)
```

```
(100, 2)
(100,)
[[ 1.77326275]
 [ 0.21477311]
 [-0.36559494]]
```

How good is the algorithm?

```
classA=[]
classB=[]
i=0
for index in y:
    if index==1.0:
        classA.append(X[i,:])
    else:
        classB.append(X[i, :])
    i+=1
a=plt.scatter(np.asarray(classA)[:,0],np.asarray(classA)[:,-1])

b=plt.scatter(np.asarray(classB)[:,0],np.asarray(classB)[:,-1])
plt.legend((a,b),('class 1','class 0'),loc='upper left')


testx = np.arange(-4.0, 4.0, 0.1)

testy = (-w[0]-w[1]*testx)/w[2]
plt.plot(testx,testy.T,color="r")

testy1 = (-w1[0]-w1[1]*testx)/w1[2]
plt.plot(testx,testy1.T,color="y")

plt.xlabel("feature 1")
plt.ylabel("feature 2")
plt.show()
```

[→]


class 1
class 0
feature 2
feature 1
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−4
−3
−2
−1
0
1
2
3
4