In [1]:
```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
import time
```

Task1

Reading datasets

In [2]:
```
A=pd.read_csv("pp3data/pp3data/A.csv",header=None)
A=A.to_numpy()
labels_A=pd.read_csv("pp3data/pp3data/labels-A.csv",header=None)
labels_A=labels_A.to_numpy().flatten()
```

In [3]:
```
B=pd.read_csv("pp3data/pp3data/B.csv",header=None)
B=B.to_numpy()
labels_B=pd.read_csv("pp3data/pp3data/labels-B.csv",header=None)
labels_B=labels_B.to_numpy().flatten()
```

In [4]:
```
USPS=pd.read_csv("pp3data/pp3data/usps.csv",header=None)
USPS=USPS.to_numpy()
labels_USPS=pd.read_csv("pp3data/pp3data/labels-usps.csv",header=None)
labels_USPS=labels_USPS.to_numpy().flatten()
```

In [5]:
```
data=[A,B,USPS]
labels=[labels_A,labels_B,labels_USPS]
```

In [6]:
```
generative=[]
discriminative=[]
average=[]
partition=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
```

Calculating generative results

In [7]:
```
for i in range(3):
    X=data[i]
    Y=labels[i]
    N = X.shape[0]
    N_test = int(N/3)
    N_train = N-N_test
    index = []
    for j in range(N):
        index.append(j)
    generative_result = {}
    for t in range(1, 31):
        test_index = random.sample(index, N_test)
        train_index = [x for x in index if x not in test_index]
        xtrain = X[train_index,:]
        ytrain = Y[train_index]
        xtest = X[test_index,:]
        ytest = Y[test_index]
        for p in partition:
            p_N = int(p*N_train)
            p_x = xtrain[:p_N,]
            p_y = ytrain[:p_N]
            class1 = p_x[(np.where(p_y==1)[0]),:]
            mu1 = np.mean(class1, axis=0)
            class2 = p_x[(np.where(p_y==0)[0]),:]
            mu2 = np.mean(class2, axis=0)
            d = X.shape[1]
            pred = np.zeros(xtest.shape[0])
            total1 = np.zeros((d,d))
            for i in range(class1.shape[0]):
                total1+= np.outer((class1[i,:]-mu1), (class1[i,:]-mu1))
            S1 = total1/class1.shape[0]
            total2 = np.zeros((d,d))
            for i in range(class2.shape[0]):
                total2+= np.outer((class2[i,:]-mu2), (class2[i,:]-mu2))
            S2 = total2/class2.shape[0]
            S = ((class1.shape[0]/p_N)*S1) + ((class2.shape[0]/p_N)*S2)
            PC1 = class1.shape[0]/p_N
            PC2 = class2.shape[0]/p_N
            W = np.dot(np.linalg.inv(S),(mu1-mu2))
            W0=np.log(PC1/PC2)+0.5*np.dot(np.dot(mu2, np.linalg.inv(S)),mu2)-
            for i in range(pred.size):
                x_test = xtest[i,:]
                pred[i] = np.dot(W,x_test) + W0
            prediction = np.where(pred>=0,1,0)
            error = (np.sum(np.logical_xor(ytest, prediction)))/ytest.size
            generative_result[(str(t),str(p))] = error
    generative.append(generative_result)
```

Calculating discriminative results

In [8]:

```
for i in range(3):
    X=data[i]
    Y=labels[i]
    N = X.shape[0]
    N_test = int(N/3)
    N_train = N-N_test
    index=[]
    for j in range(N):
        index.append(j)
    discriminative_result = {}
    logisticR_X = np.c_[np.ones(N),X]
    for times in range(1, 31):
        test_index = random.sample(index, N_test)
        train_index = [x for x in index if x not in test_index]
        xtrain = logisticR_X[train_index,:]
        ytrain = Y[train_index]
        xtest = logisticR_X[test_index,:]
        ytest = Y[test_index]
        for pv in partition:
            pv_N = int(pv*N_train)
            p_x = xtrain[:pv_N,]
            p_y = ytrain[:pv_N]
            W_old = np.zeros(p_x.shape[1])
            W_new = np.matrix(np.zeros(p_x.shape[1]))
            d = X.shape[1]
            for iteration in range(100):
                if not(np.linalg.norm(W_new-W_old)**2)/(np.linalg.norm(W_old)
                      break
                a = np.dot(p_x, W_old)
                y = 1/(1 + np.exp(-a))
                R = np.diag(y*(1-y))
                W_new = W_old - np.dot(np.linalg.inv(0.1*np.identity(d+1) + r
                W_old=W_new
            a = np.dot(p_x, W_new)
            y = 1/(1 + np.exp(-a))
            total = np.zeros((d+1,d+1))
            for i in range(y.size):
                total+= (y[i]*(1-y[i]))*np.dot(p_x[i,:], p_x[i,:])
            SN = 0.1*np.identity(d+1) + total
            prob = np.zeros(xtest.shape[0])
            for i in range(prob.size):
                phi = xtest[i,:]
                mu = np.dot(W_new.T, phi)
                sigma_square = np.dot(np.dot(phi.T, SN), phi)
                k = 1/np.sqrt(1 + (np.pi*sigma_square/8))

                prob[i] = 1/(1 + np.exp(-(k*mu)))

            pred_labels = np.where(prob>=0.5,1,0)
            misclassified_instances = np.sum(np.logical_xor(ytest, pred_label
            test_error_rate = misclassified_instances/ytest.size


            discriminative_result[(str(times),str(pv))] = test_error_rate

    discriminative.append(discriminative_result)
```

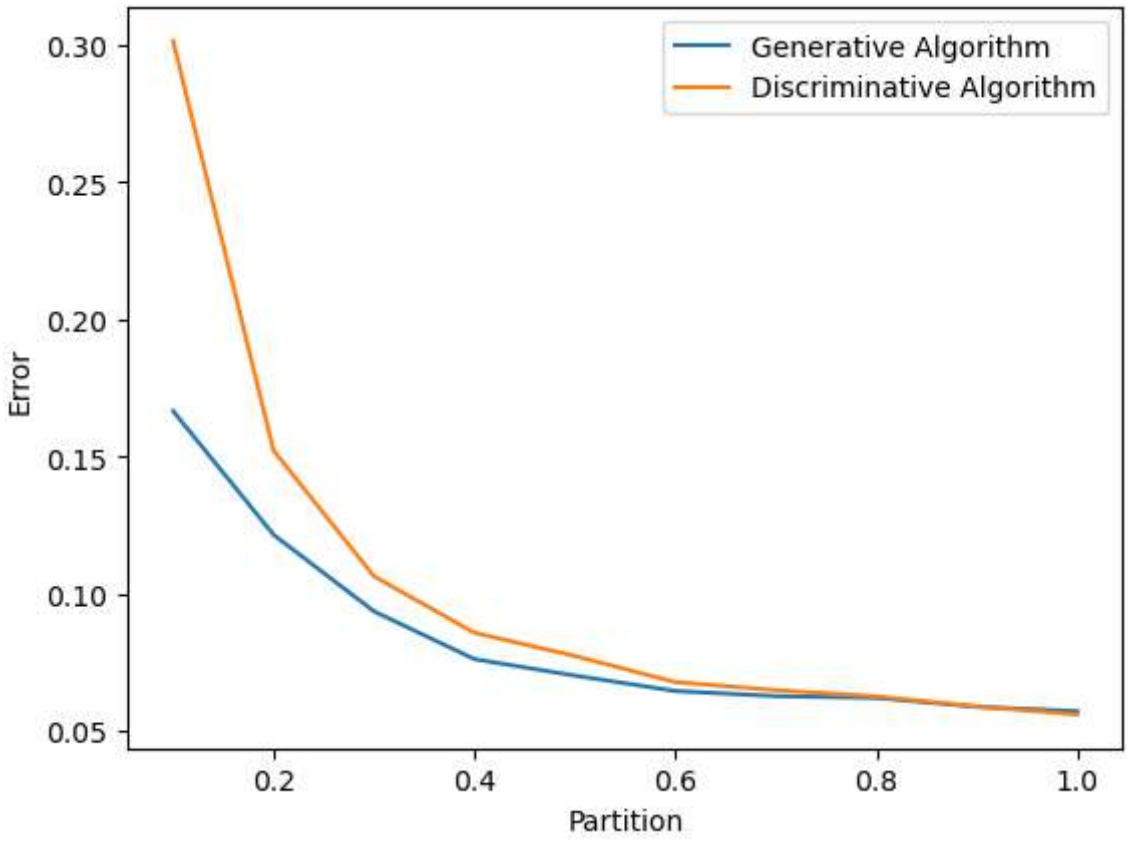
```
C:\Users\Aaryan Agarwal\AppData\Local\Temp\ipykernel_23140\1645444686.py:2
7: RuntimeWarning: invalid value encountered in true_divide
  if not(np.linalg.norm(W_new-W_old)**2)/(np.linalg.norm(W_old)**2>0.001):
```

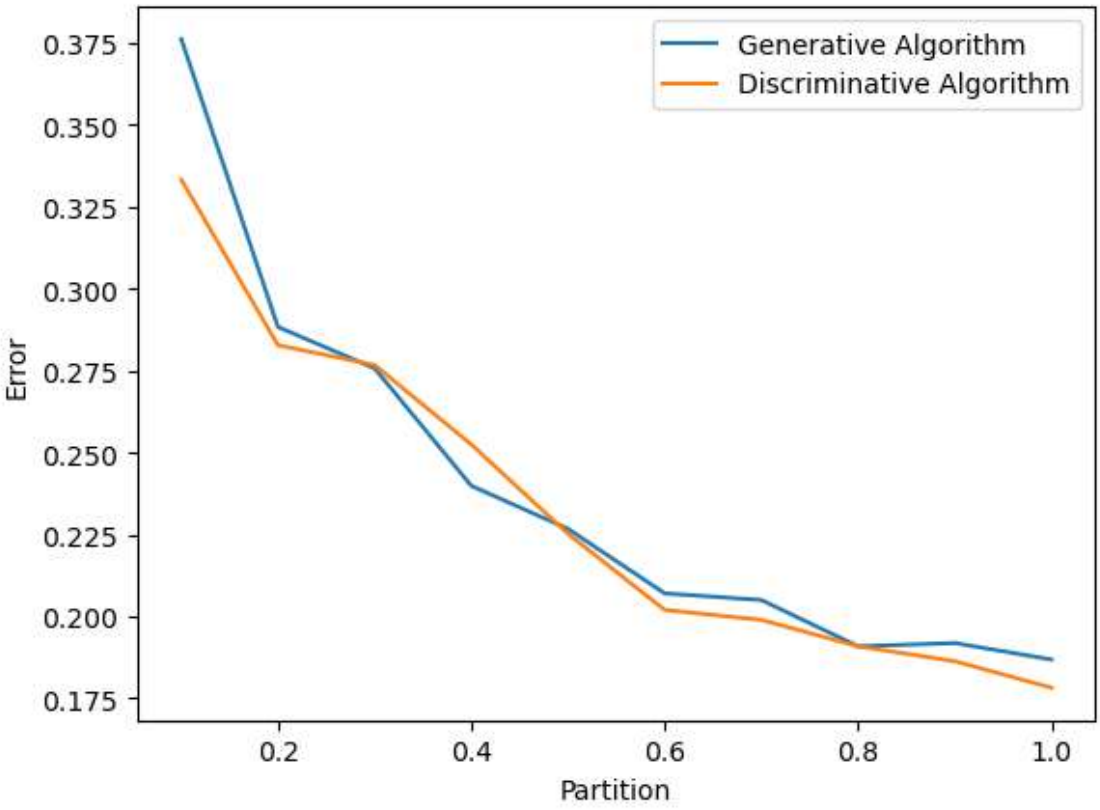
Calculating average

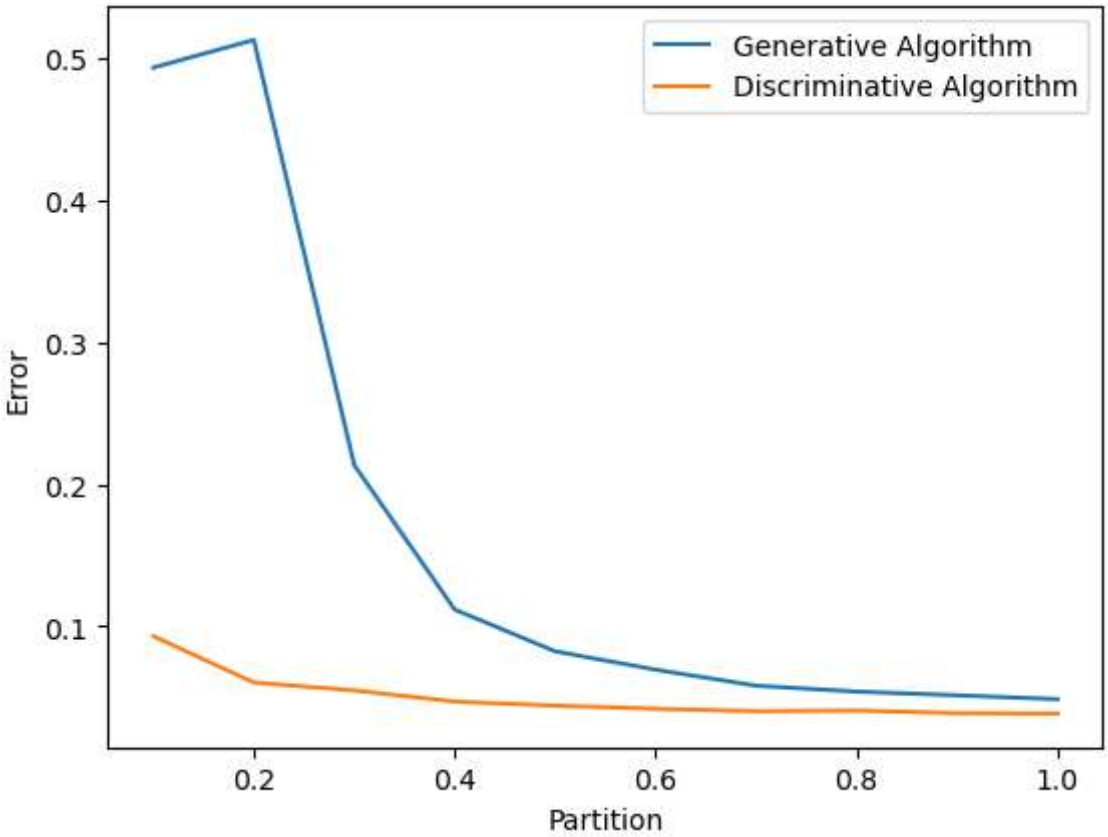
In [9]:
```
average=[]
for i in range(3):
    g_avg = {}
    d_avg = {}

    for p in partition:
            x = np.array([generative[i][(str(t),str(p))] for t in range(1,31)
            y = np.array([discriminative[i][(str(t),str(p))] for t in range(1
            g_avg[(str(p), "Generative")] = (np.mean(x), np.std(x))
            d_avg[(str(p), "Discriminative")] = (np.mean(y), np.std(y))
    average.append ((g_avg, d_avg))
```

Graphs

In [13]:
```
for i in range(3):
    plt.plot(partition, [average[i][0][(str(p),'Generative')][0] for p in par
    plt.plot(partition, [average[i][1][(str(p),'Discriminative')][0] for p in
    plt.xlabel('Partition')
    plt.ylabel('Error')
    plt.legend()
    plt.show()
```




Generative Algorithm
Discriminative Algorithm
Error
Partition
0.375
0.350
0.325
0.300
0.275
0.250
0.225
0.200
0.175
0.2
0.4
0.6
0.8
1.0


Generative Algorithm
Discriminative Algorithm
Error
Partition
0.5
0.4
0.3
0.2
0.1
0.2
0.4
0.6
0.8
1.0

Task2

In [7]:
```
newdata=[data[0],data[2]]
newlabel=[labels[0],labels[2]]
newton=[]
gradient=[]
```

In [8]:
```
for i in range(1):
    new_d=np.c_[np.ones(newdata[i].shape[0]),newdata[i]]
    N = new_d.shape[0]
    train_size = int((2/3) * len(newdata[i]))
    xtrain = new_d[:train_size]
    ytrain = newlabel[0][:train_size]
    xtest = new_d[train_size:]
    ytest = newlabel[0][train_size:]
    W_old=np.zeros(xtrain.shape[1])
    weight_vector = {}
    errors = {}
    for iteration in range(6000):
        d = new_d.shape[1]
        a = np.dot(xtrain, W_old)
        y = 1/(1 + 1/np.exp(a))
        R = np.diag(y*(1-y))
        W_new = W_old - np.dot(np.linalg.inv(0.1*np.identity(d) + np.dot(np.d
        ts = time.time()
        weight_vector[ts] = W_old
        if ((np.linalg.norm(W_new-W_old)**2)/(np.linalg.norm(W_old)**2)<0.001
            break
        else:
            W_old = W_new
            a = np.dot(xtrain, W_new)
            y = 1/(1 + np.exp(-a))
            total = np.zeros((d,d))
            for i in range(y.size):
                total+= (y[i]*(1-y[i]))*np.dot(xtrain[i,:], xtrain[i,:])
            SN = 0.1*np.identity(d) + total
            pred = np.zeros(xtest.shape[0])
            for i in range(pred.size):
                phi = xtest[i,:]
                mu = np.dot(W_new.T, phi)
                sigma_sq = np.dot(np.dot(phi.T, SN), phi)
                a = 1/np.sqrt(1 + (np.pi*sigma_sq/8))
                pred[i] = 1/(1 + np.exp(-(a*mu)))
            prediction = np.where(pred>=0.5,1,0)
            error = np.sum(np.logical_xor(ytest, prediction))
            test_error = error/ytest.size
            errors[ts] = test_error
    newton.append((errors,weight_vector))
```

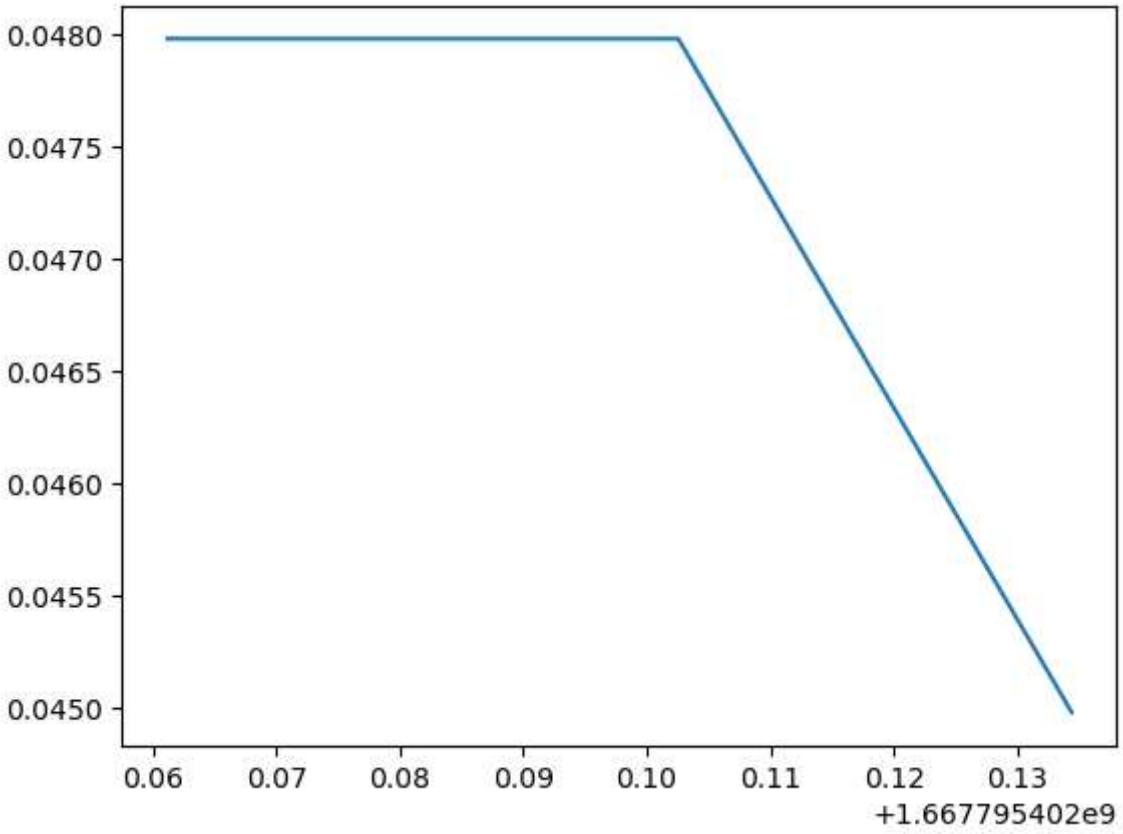
```
C:\Users\Aaryan Agarwal\AppData\Local\Temp\ipykernel_29020\1619481033.py:2
0: RuntimeWarning: divide by zero encountered in double_scalars
  if ((np.linalg.norm(W_new-W_old)**2)/(np.linalg.norm(W_old)**2)<0.001):
```

In [11]:

```
for i in range(1):
    new_d=np.c_[np.ones(newdata[i].shape[0]),newdata[i]]
    N = new_d.shape[0]
    d = new_d.shape[1]
    train_size = int((2/3) * len(newdata[i]))
    xtrain = new_d[:train_size]
    ytrain = newlabel[0][:train_size]
    xtest = new_d[train_size:]
    ytest = newlabel[0][train_size:]
    W_old = np.zeros(xtrain.shape[1])
    weight_vector = {}
    errors = {}
    for iteration in range(6000):
        a = np.dot(xtrain, W_old)
        y = 1/(1 + 1/np.exp(a))
        term_1 = np.dot(xtrain.T, (y-ytrain)) + 0.1*W_old
        W_new = W_old - 0.001*(term_1)
        if(iteration%10 == 0):
            ts = time.time()
            weight_vector[ts] = W_old
        if ((np.linalg.norm(W_new-W_old)**2)/(np.linalg.norm(W_old)**2)<0.001
            break
        else:
            W_old = W_new
            # SN calculation
            a = np.dot(xtrain, W_new)
            y = 1/(1 + np.exp(-a))
            summation = np.zeros((d,d))
            for i in range(y.size):
                summation+= (y[i]*(1-y[i]))*np.dot(xtrain[i,:], xtrain[i,:])

            SN = 0.1*np.identity(d) + summation
            #predictions

            pred = np.zeros(xtest.shape[0])
            for i in range(pred.size):
                phi = xtest[i,:]
                mu = np.dot(W_new.T, phi)
                sigma_sq = np.dot(np.dot(phi.T, SN), phi)
                a = 1/np.sqrt(1 + (np.pi*sigma_sq/8))
                pred[i] = 1/(1 + np.exp(-(a*mu)))

            prediction = np.where(pred>=0.5,1,0)
            error = np.sum(np.logical_xor(ytest, prediction))
            test_error = error/ytest.size
            errors[ts] = test_error
    gradient.append((errors,weight_vector))
```

```
C:\Users\Aaryan Agarwal\AppData\Local\Temp\ipykernel_29020\107682710.py:21:
RuntimeWarning: divide by zero encountered in double_scalars
  if ((np.linalg.norm(W_new-W_old)**2)/(np.linalg.norm(W_old)**2)<0.001):
```

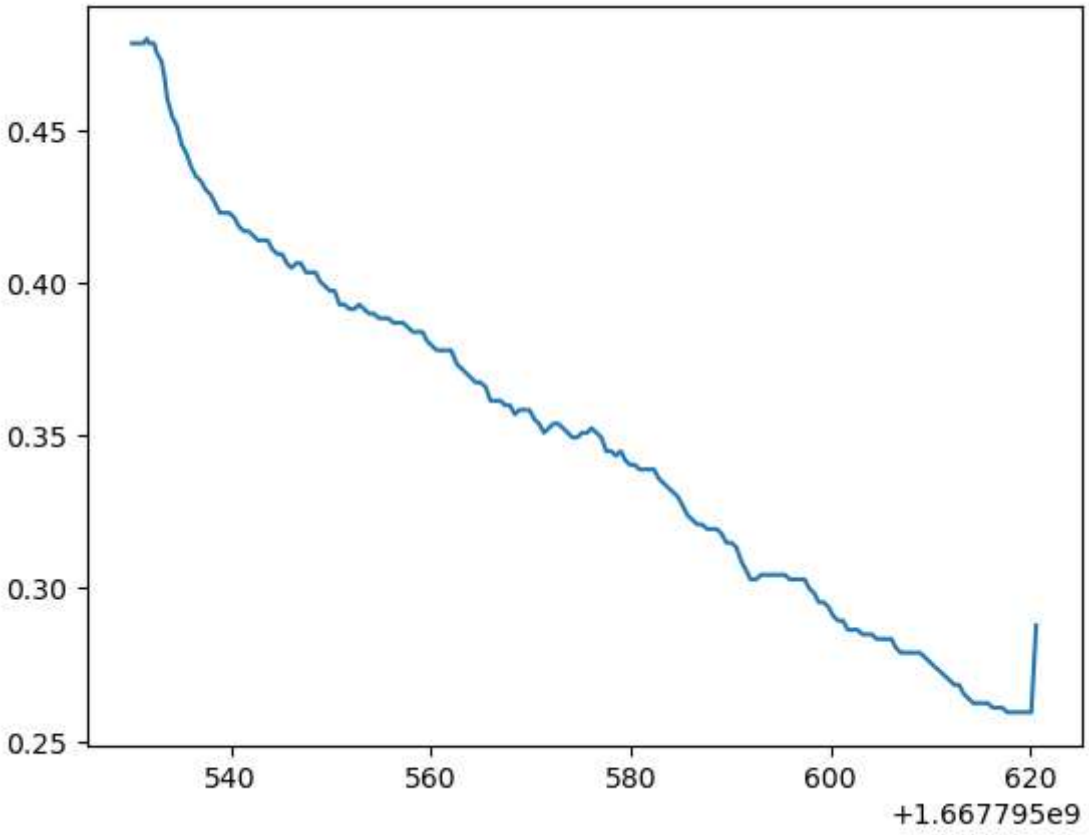
In [10]:
```
plt.plot(newton[0][0].keys(),newton[0][0].values())
plt.show()
```



In [14]:
```
plt.plot(newton[1][0].keys(),newton[1][0].values())
plt.show()
```

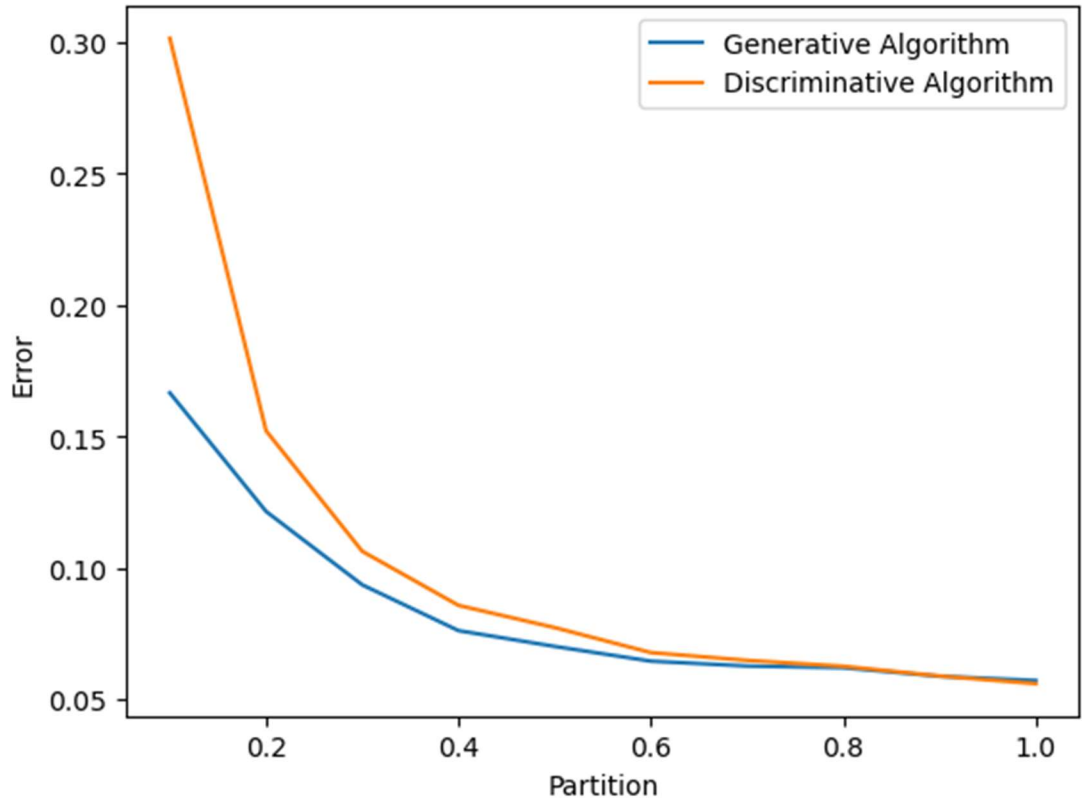
```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_9364\2307579251.py in <module>
----> 1 plt.plot(newton[1][0].keys(),newton[1][0].values())
      2 plt.show()

IndexError: list index out of range
```

In [12]:
```
plt.plot(gradient[1][0].keys(),gradient[1][0].values())
plt.show()
```

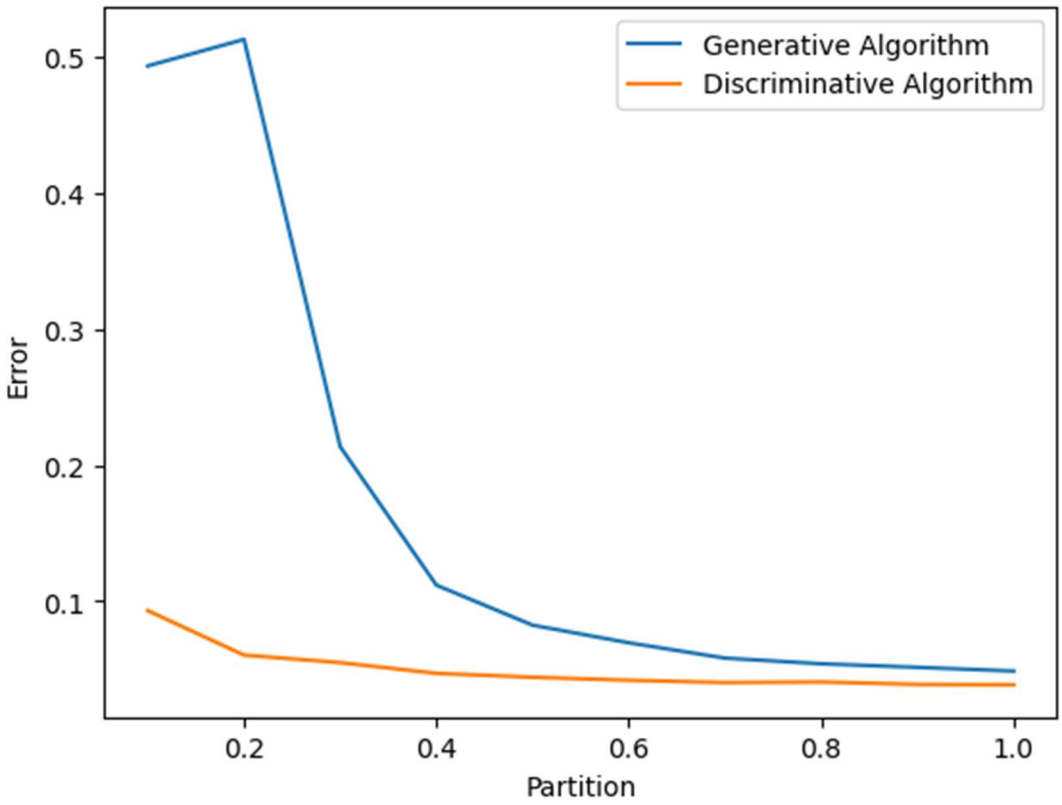


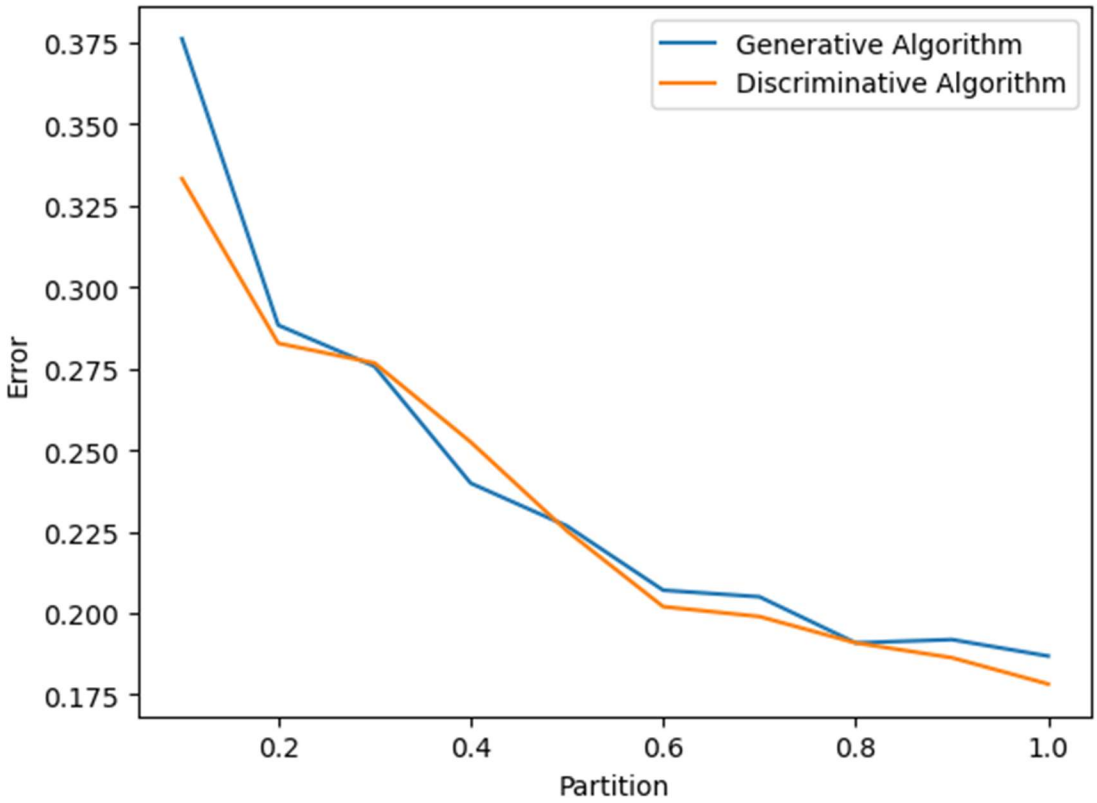
In [ ]:
```

```

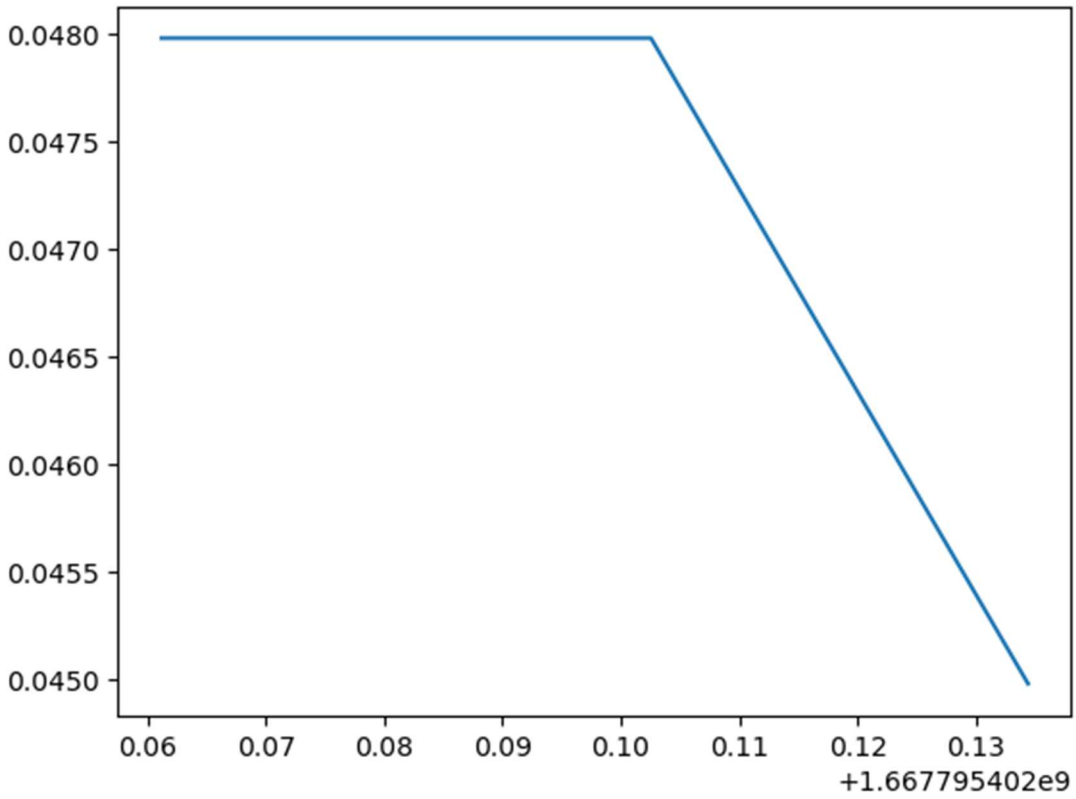
Assignment 3

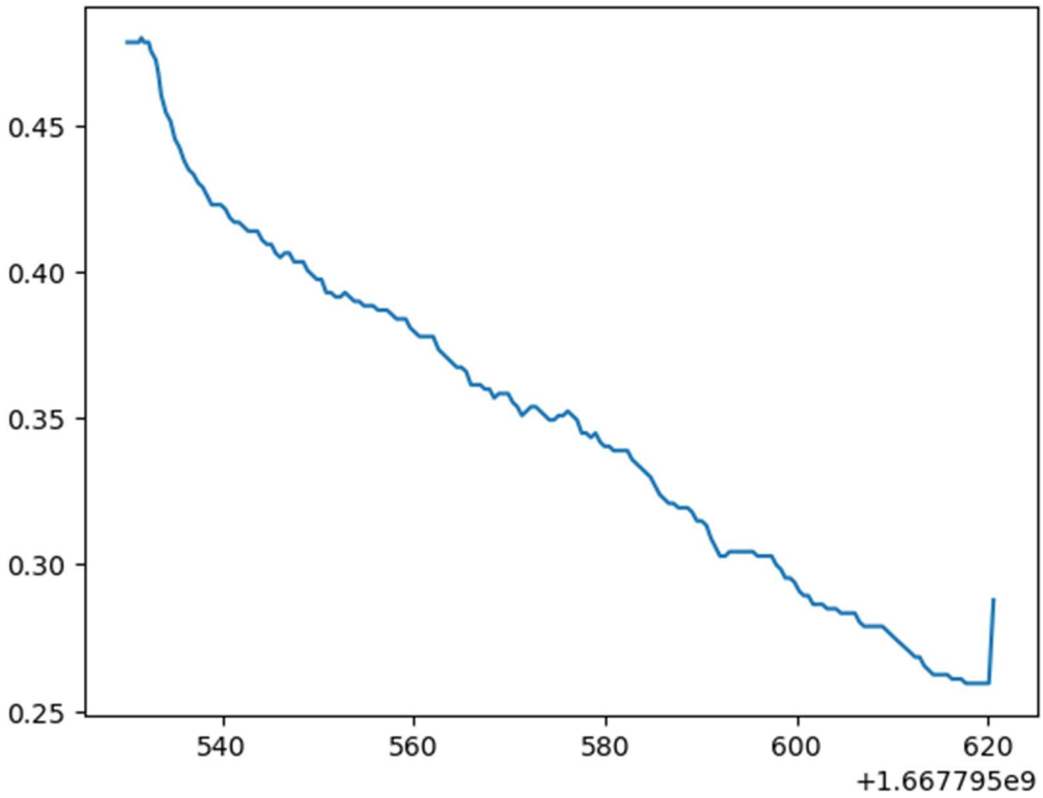


We can conclude that generative performed better when data set is small and discriminative performed better when data set is bigger


Generative Algorithm
Discriminative Algorithm
0.5
0.4
0.3
0.2
0.1
Error
0.2
0.4
0.6
0.8
1.0
Partition


Generative Algorithm
Discriminative Algorithm
0.375
0.350
0.325
0.300
0.275
0.250
0.225
0.200
0.175
Error
0.2
0.4
0.6
0.8
1.0
Partition


0.0480
0.0475
0.0470
0.0465
0.0460
0.0455
0.0450
0.06
0.07
0.08
0.09
0.10
0.11
0.12
0.13
+1.667795402e9


0.45
0.40
0.35
0.30
0.25
540
560
580
600
620
+1.667795e9

For the dataset A newton performed better than gradient ascent. The USPS dataset was not working for my code so couldn’t derive any conclusion from that

The assignment is written in ipynb format and can be run using the jupyter notebook

The data files are attached to the directory, in case of data not loading please check the path

There are no specific instructions to run the file