Dimensionality Reduction and Visualization

Exercise set 1 Solutions

Md. Abdullah-Al Mamun

In [1]:

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

In [ ]:

```
#Problem Pre1:

def DataPointGenerate(dimens):
    datapoints = 10000000
    ar = np.empty(shape=(dimens,datapoints))
    for x in range(dimens):
        ar[x] = np.random.uniform(1,-1,datapoints)
        outsideHypershpere=0
        insideHypersphere=0
        insideHyperSphericalShell=0
    for x in range(datapoints):
        distance = np.linalg.norm(ar[:,x])
        if distance <= 1 :
            insideHypersphere  += 1
        if 0.95<distance and distance<1:
            insideHyperSphericalShell+=1
        else:
            outsideHypershpere+=1


    print ("Dimensensions: " + str(dimens),str(float(insideHypersphere)/float(points
    print ("Dimensensions: " + str(dimens), str(float(insideHyperSphericalShell) / f




for y in [2,3,5,7,10,13,17]:
    DataPointGenerate(y)
```

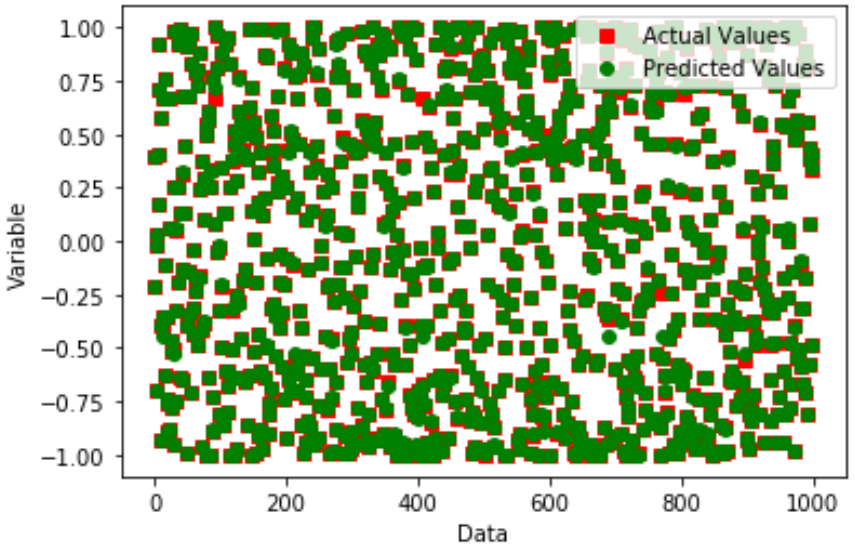
In [8]:

```
#Problem Pre2

def GenerateDataPoints(dim):
    points=2000
    dataset=[]
    data = np.empty(shape=(dim,points))
    for i in range(dim):
        data[i] = np.random.normal(0,1,points)

    for i in data[0]:
        target = np.sin(i)
        t = (i,target)
        dataset.append(t)

    training_data = data[:,:int(points/2)]
    testing_data = data[:,int(points/2):int(points)]
    return training_data,testing_data,points,dataset
```

In [9]:

```
def getNeighbors(training_data,testing_data,k,points):
    k=5
    Euclidean = []
    Neighbors=[]
    Neighbors_x_test=[]
    for x in range(0,int(points/2)):
        dist=(training_data[0][x],np.linalg.norm(training_data[:,x]-testing_data))
        Euclidean.append(dist)
        Euclidean.sort(key=operator.itemgetter(1))
    for i in range(k):
        Neighbors.append(Euclidean[i][0])
    return testing_data[0],Neighbors

def Predict(Neighbors,x_test,dataset):
    sum=0
    x_test = x_test
    for i in Neighbors:
        for j in dataset:
            if i == j[0]:
                result = sum + j[1]
                sum = result
                break

    predicted = float(result)/float(len(Neighbors))

    for i in dataset:
        if x_test == i[0]:
            target_actual = i[1]
            break
    return predicted,target_actual

def Visualization(pred_actual):
    plt.xlabel("Data")
    plt.ylabel("Variable")
    plt.plot(list(zip(*pred_actual))[2],'rs' ,label="Actual Values")
    plt.plot(list(zip(*pred_actual))[1],'go',label="Predicted Values")
    plt.legend(loc='upper right',ncol=1,fontsize=10)
    plt.show()

def Error(pred_actual,points):
    SSE=0
    actual = list(zip(*pred_actual))[2]
    pred = list(zip(*pred_actual))[1]
    for x in range(len(pred_actual)):
        SSE += (actual[x] - pred[x])**2
    return float(float(SSE)/1000)

def Main():

    pred_actual=[]

    training_data,testing_data,points,dataset = GenerateDataPoints(1)

# Similary we can use datapoints for d = 2, d = 4, d = 7, d = 10, d = 15
```

```
    for x in range(0,int(points/2)):
        x_test,Neighbors = getNeighbors(training_data,testing_data[:,x],5,points)
        Predicted,Actual=Predict(Neighbors,x_test, dataset)
        result = (x_test,Predicted,Actual)
        pred_actual.append(result)
    Visualization(pred_actual)
    print("Sum of Squared error : " + str(Error(pred_actual,points)))


Main()
```




```
Sum of Squared error : 0.0012873233909209 3
```