assignmentThree

December 6, 2022

```
[ ]: from sklearn.datasets import make_circles
     from sklearn.model_selection import train_test_split
     from sklearn.utils import shuffle
     from sklearn.preprocessing import OneHotEncoder
     from sklearn.metrics import accuracy_score
     from sklearn.tree import DecisionTreeClassifier

     from collections import Counter
     from collections import OrderedDict

     import numpy as np
     import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt

     from tensorflow.keras.models import Sequential
     from keras.utils.vis_utils import plot_model
     from tensorflow.keras.layers import Dense
```

```
/Users/nick/miniforge3/lib/python3.9/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version
of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

```
[ ]: SAMPLESIZE = 500
     TESTSIZE = 0.90
     VALSIZE = 0.70
```

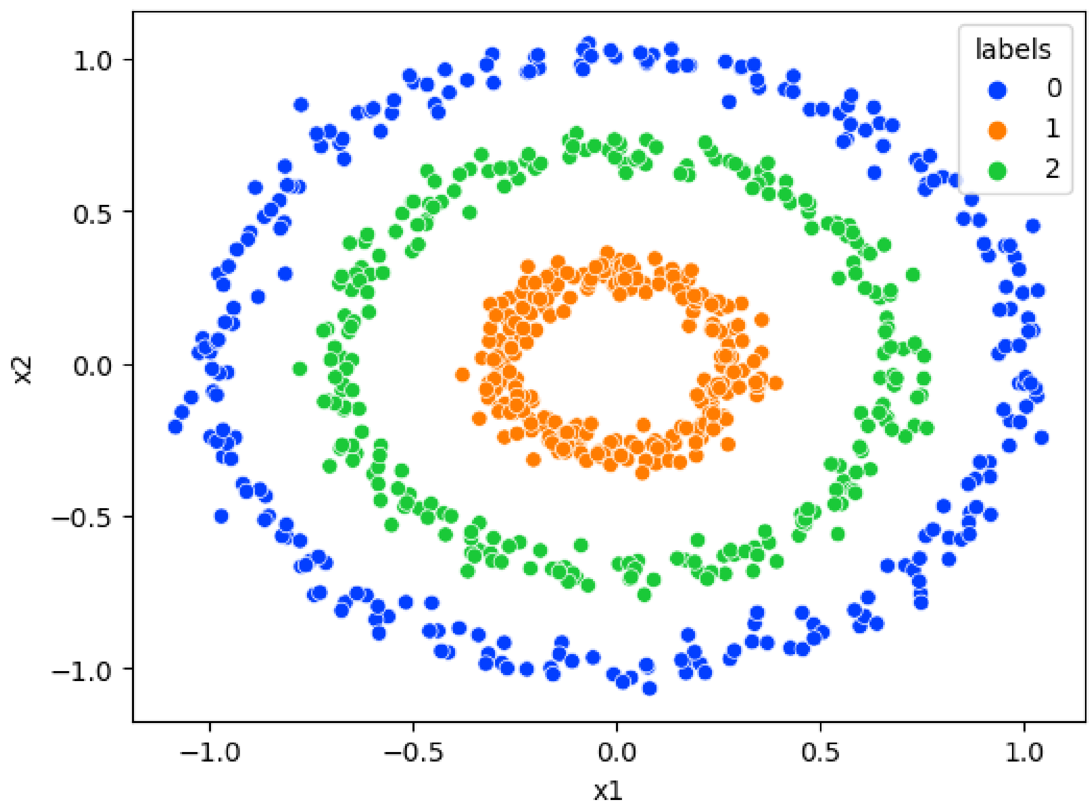
0.1 Programming with Supervised Learning

0.1.1 Generating Data One

```
[ ]: X_small, y_small = make_circles(n_samples=(int(SAMPLESIZE/2),int(SAMPLESIZE/
      ↪2)), random_state=3,
     noise=0.04, factor = 0.3)
     X_large, y_large = make_circles(n_samples=(int(SAMPLESIZE/2),int(SAMPLESIZE/
      ↪2)), random_state=3,
     noise=0.04, factor = 0.7)
```

```
y_large[y_large==1] = 2

df = pd.DataFrame(np.vstack([X_small,X_large]),columns=['x1','x2'])
df['labels'] = np.hstack([y_small,y_large])
dfOne = shuffle(df)
trainOne, valTestOne = train_test_split(dfOne, test_size=TESTSIZE)
valone, testOne = train_test_split(valTestOne, test_size=VALSIZE)


sns.scatterplot(data=df,x='x1',y='x2',hue='labels',palette="bright")
```

[ ]: <AxesSubplot:xlabel='x1', ylabel='x2'>

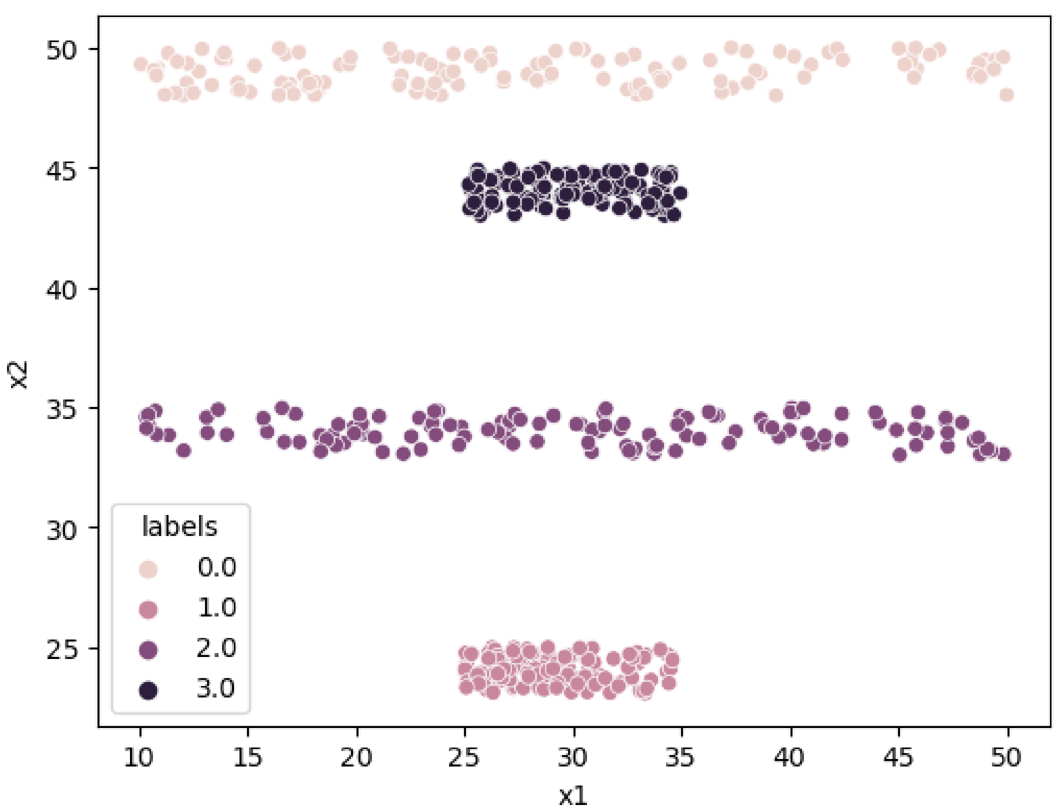


0.1.2 Generating Data Five

[ ]:
```
X = []
y = []
size = int(SAMPLESIZE/4)

X.extend(list(np.random.uniform(low=10, high=50, size=(size,))))
y.extend(list(np.random.uniform(low=48, high=50, size=(size,))))
```

```
c = list(np.zeros(size))

X.extend(list(np.random.uniform(low=25, high=35, size=(size,))))
y.extend(list(np.random.uniform(low=23, high=25, size=(size,))))
c.extend(np.ones(size))

X.extend(list(np.random.uniform(low=10, high=50, size=(size,))))
y.extend(list(np.random.uniform(low=33, high=35, size=(size,))))
c.extend(np.ones(size)*2)

X.extend(list(np.random.uniform(low=25, high=35, size=(size,))) )
y.extend(list(np.random.uniform(low=43, high=45, size=(size,))))
c.extend(np.ones(size)*3)


dfTwo = pd.DataFrame(data={'x1': X, 'x2': y,'labels':c})
dfTwo = shuffle(dfTwo)

trainTwo, valTestTwo = train_test_split(dfTwo, test_size=TESTSIZE)
valTwo, testTwo = train_test_split(valTestTwo, test_size=VALSIZE)
sns.scatterplot(data=dfTwo,x='x1',y='x2',hue='labels')
```

[ ]: <AxesSubplot:xlabel='x1', ylabel='x2'>

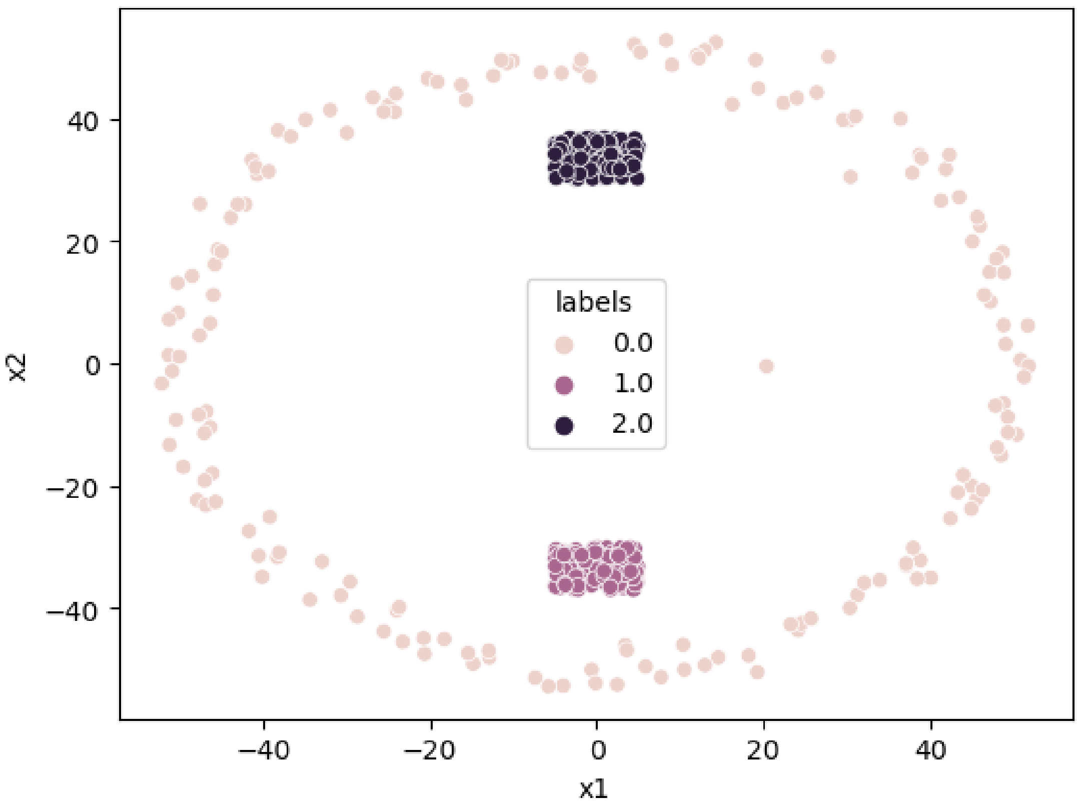


0.1.3 Generating Data Six

[ ]:
```python
size = int(SAMPLESIZE/3)
print(size)
X,y = make_circles(n_samples=(size*2), random_state=3,
noise=0.04, factor = 0.3)

X1 = list(X[:, 0].flatten()*50)
X2 = list(X[:, 1].flatten()*50)

X = []
y = []


for x in range(0,len(X1)):

    if ((((X1[x] > -20) and (X1[x] < 20))) and (((X2[x] > -20) and (X2[x] <␣
 ↪20)))):
        n=1
    else:
        X.append(X1[x])
        y.append(X2[x])

Xc = list(np.zeros(len(X)))
Xc.pop
X.extend(list(np.random.uniform(low=-5, high=5, size=(size,))))
y.extend(list(np.random.uniform(low=-37, high=-30, size=(size,))))
Xc.extend(list(np.ones(size)))

X.extend(list(np.random.uniform(low=-5, high=5, size=(size,))))
y.extend(list(np.random.uniform(low=30, high=37, size=(size,))))
Xc.extend(list(np.ones(size)*2))

print(len( X))
print(len( y))
print(len( Xc))
dfThree= pd.DataFrame(data={'x1': X, 'x2': y,'labels':Xc})
sns.scatterplot(data=dfThree,x='x1',y='x2',hue='labels')
```

```
166
499
499
499
```

[ ]:
```
<AxesSubplot:xlabel='x1', ylabel='x2'>
```




```
[ ]: dfThree = shuffle(dfThree)

     trainThree, valTestThree = train_test_split(dfThree, test_size=TESTSIZE)
     valThree, testThree = train_test_split(valTestThree, test_size=VALSIZE)
```

0.1.4 KNN Programmed

```
[ ]: #KNN implementation
     def KNN(X,y,k,labels,Xtest,ytest):

         X = list(X)
         y = list(y)
         Xtest = list(Xtest)
         ytest= list(ytest)
         labels = list(labels)

         d = OrderedDict()
         preds = {}

         #Go through test examples
         for index in range(0,len(Xtest)):
```

```
        #Go through train examples
        for index2 in range(0,len(X)):

            #Ensure not same value
            if (index2 != index):

                #Compute euclediian distance - save this value to a dictionary␣
↪with index
                d[str(index2)] = np.sqrt((Xtest[index]- X[index2])**2 +␣
↪(ytest[index] - y[index2])**2)

        #Sort the euclediian distances
        sortedDistances = dict(sorted(d.items(), key=lambda item: item[1]))


        votes = []

        #Look at K nearest neighbor labels that are in the training data
        #Collect K nearest labels
        for number in range(0,k):
            votes.append(labels[int(list(sortedDistances.keys())[number])])

        #Set the label to the most "voted" label
        preds[str(index)] = Counter(votes).most_common(1)[0][0]
    return preds
```

[ ]:
```
#Manually coded accuracy function
def checkAccuracy(pred,test):
    wrong = 0

    for idx in range(len(pred)):
        if (pred[idx] != test[idx]):
            wrong = wrong + 1
    return 1- wrong/len(pred)
```

[ ]:
```
#Function to check for best k value in an array of different k values for KNN
def bestK(dfTrain,dfTest):


    k = [1,3,5,7,9]
    accuracies = []

    #Try each k value
    for x in k:

        #Predict labels with KNN with the K
```

```
        predicted = KNN(dfTrain.x1,dfTrain.x2,x,dfTrain.labels,dfTest.x1,dfTest.
↪x2)

        #Save accuracy of this prediction
        accuracies.append(checkAccuracy(list(predicted.values()),list(dfTest.
↪labels)))


    print("Best K: ", k[np.argmax(accuracies)])

    #Return the K value that gives the best accuracy tested
    return k[np.argmax(accuracies)]
```

[ ]:
```
def KNNAccuracy(train,valid,test):

    k = bestK(train,valid)
    predicted = KNN(train.x1,train.x2,k,train.labels,valid.x1,valid.x2)
    print("\nValidation Accuracy", checkAccuracy(list(predicted.
↪values()),list(valid.labels)))
    predicted = KNN(train.x1,train.x2,k,train.labels,test.x1,test.x2)
    print("Test Accuracy", checkAccuracy(list(predicted.values()),list(test.
↪labels)))
```

0.1.5 KNN on Data

[ ]:
```
print("Dataset One")
KNNAccuracy(trainOne,valone,testOne)

print("\nDataset Two")
KNNAccuracy(trainTwo,valTwo,testTwo)

print("\nDataset Three")
KNNAccuracy(trainThree,valThree,testThree)
```

```
Dataset One
Best K:  1

Validation Accuracy 0.9814814814814815
Test Accuracy 0.9904761904761905

Dataset Two
Best K:  1

Validation Accuracy 1.0
Test Accuracy 1.0

Dataset Three
Best K:  3
```

```
Validation Accuracy 0.9037037037037037
Test Accuracy 0.9111111111111111
```

0.2 Comparison of K-NN and Decision Tree

0.2.1 Coding Decision Tree with Existing Frameworks

[ ]:
```python
#Function to return accuracy of decision tree on data
def decisionTree(train,val,test):

    #Create decision tree
    clf = DecisionTreeClassifier(max_depth = 2)
    clf.fit(X=np.array(train.x1,train.x2).reshape(-1, 1),y=np.array(train.
 ↪labels).reshape(-1, 1))

    y_pred = clf.predict(np.array(val.x1,val.x2).reshape(-1, 1))
    print("Validation Accuracy", accuracy_score(np.array(val.labels).
 ↪reshape(-1, 1), y_pred))

    y_pred = clf.predict(np.array(test.x1,test.x2).reshape(-1, 1))
    print("Test Accuracy",accuracy_score(np.array(test.labels).reshape(-1, 1),␣
 ↪y_pred))
```

0.2.2 Decision Tree on Data vs K-NN

[ ]:
```python
print("Dataset One")
print("\nDecision Tree")
decisionTree(trainOne,valone,testOne)

print("\nKNN")
KNNAccuracy(trainOne,valone,testOne)

print("\nDataset Two")
print("\nDecision Tree")
decisionTree(trainTwo,valTwo,testTwo)

print("\nKNN")
KNNAccuracy(trainTwo,valTwo,testTwo)

print("\nDataset Three")
print("\nDecision Tree")
decisionTree(trainThree,valThree,testThree)

print("\nKNN")
KNNAccuracy(trainThree,valThree,testThree)
```

```
Dataset One
```

```
Decision Tree
Validation Accuracy 0.5481481481481482
Test Accuracy 0.5603174603174603

KNN
Best K:  1

Validation Accuracy 0.9814814814814815
Test Accuracy 0.9904761904761905

Dataset Two

Decision Tree
Validation Accuracy 0.4148148148148148
Test Accuracy 0.4444444444444444

KNN
Best K:  1

Validation Accuracy 1.0
Test Accuracy 1.0

Dataset Three

Decision Tree
Validation Accuracy 0.5925925925925926
Test Accuracy 0.6

KNN
Best K:  3

Validation Accuracy 0.9037037037037037
Test Accuracy 0.9111111111111111
```

0.3 Deep Neural Networks

0.3.1 Deep Neural Network Coding

[ ]:
```
def runModelOne(trainData,valData,testData):

    enc = OneHotEncoder()
    labels = np.array(trainData.labels).reshape(-1, 1)
    enc.fit(labels)
    ytrain = enc.transform(labels).toarray()
    Xtrain = np.array([trainData.x1,trainData.x2]).T
    Xtest = np.array([testData.x1,testData.x2]).T
    XVal = np.array([valData.x1,valData.x2]).T
```

```
    encTwo = OneHotEncoder()
    labels = np.array(testData.labels).reshape(-1, 1)
    encTwo.fit(labels)
    ytest = encTwo.transform(labels).toarray()


    encThree = OneHotEncoder()
    labels = np.array(valData.labels).reshape(-1, 1)
    encThree.fit(labels)
    yval = encThree.transform(labels).toarray()

    model = Sequential()
    model.add(Dense(20, input_shape=(2,), activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(len(ytrain[0]), activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
    model.fit(Xtrain, ytrain, epochs=160, batch_size=10,verbose=0)

    _, accuracy = model.evaluate(XVal, yval)
    print('Validation Accuracy: %.2f' % (accuracy*100))

    _, accuracy = model.evaluate(Xtest, ytest)
    print('Test Accuracy: %.2f' % (accuracy*100))

#plot_model(model, to_file='model_plot.png', show_shapes=True,␣
 ↪show_layer_names=True)
```

[ ]:
```
def runModelTwo(trainData,valData,testData):

    enc = OneHotEncoder()
    labels = np.array(trainData.labels).reshape(-1, 1)
    enc.fit(labels)
    ytrain = enc.transform(labels).toarray()
    Xtrain = np.array([trainData.x1,trainData.x2]).T
    Xtest = np.array([testData.x1,testData.x2]).T
    XVal = np.array([valData.x1,valData.x2]).T

    encTwo = OneHotEncoder()
    labels = np.array(testData.labels).reshape(-1, 1)
    encTwo.fit(labels)
    ytest = encTwo.transform(labels).toarray()


    encThree = OneHotEncoder()
    labels = np.array(valData.labels).reshape(-1, 1)
```

```
    encThree.fit(labels)
    yval = encThree.transform(labels).toarray()

    model = Sequential()
    model.add(Dense(20, input_shape=(2,), activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(len(ytrain[0]), activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
    model.fit(Xtrain, ytrain, epochs=160, batch_size=10,verbose=0)

    _, accuracy = model.evaluate(XVal, yval)
    print('Validation Accuracy: %.2f' % (accuracy*100))

    _, accuracy = model.evaluate(Xtest, ytest)
    print('Test Accuracy: %.2f' % (accuracy*100))

    #plot_model(model, to_file='model_plot.png', show_shapes=True,␣
↪show_layer_names=True)
```

0.3.2 Deep Neural Networks Performance on Data

Model One

[ ]:
```
runModelOne(trainOne,valone, testOne)
```

```
2022-12-06 21:22:34.539181: W
tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU
frequency: 0 Hz
```

```
9/9 [==============================] - 0s 681us/step - loss: 0.0680 - accuracy:
1.0000
Validation Accuracy: 100.00
20/20 [==============================] - 0s 540us/step - loss: 0.0678 -
accuracy: 0.9968
Test Accuracy: 99.68
```

[ ]:
```
runModelOne(trainTwo,valTwo,testTwo)
```

```
5/5 [==============================] - 0s 834us/step - loss: 1.1932 - accuracy:
0.4593
Validation Accuracy: 45.93
10/10 [==============================] - 0s 592us/step - loss: 1.2024 -
accuracy: 0.4984
Test Accuracy: 49.84
```

[ ]:
```
runModelOne(trainThree,valThree,testThree)
```

```
5/5 [==============================] - 0s 807us/step - loss: 0.6846 - accuracy:
0.8370
Validation Accuracy: 83.70
10/10 [==============================] - 0s 641us/step - loss: 0.4628 -
accuracy: 0.8952
Test Accuracy: 89.52
```

Model Two

```
[ ]: runModelTwo(trainOne,valone, testOne)
```

```
9/9 [==============================] - 0s 627us/step - loss: 0.0041 - accuracy:
1.0000
Validation Accuracy: 100.00
20/20 [==============================] - 0s 549us/step - loss: 0.0038 -
accuracy: 1.0000
Test Accuracy: 100.00
```

```
[ ]: runModelTwo(trainTwo,valTwo,testTwo)
```

```
5/5 [==============================] - 0s 805us/step - loss: 0.9410 - accuracy:
0.4741
Validation Accuracy: 47.41
10/10 [==============================] - 0s 681us/step - loss: 0.9194 -
accuracy: 0.5111
Test Accuracy: 51.11
```

```
[ ]: runModelTwo(trainThree,valThree,testThree)
```

```
5/5 [==============================] - 0s 874us/step - loss: 1.0131 - accuracy:
0.8741
Validation Accuracy: 87.41
10/10 [==============================] - 0s 754us/step - loss: 0.7924 -
accuracy: 0.9079
Test Accuracy: 90.79
```

0.4 Noisy Data Generation

0.4.1 Different Noise Levels

I chose to create Gaussian noise here. I define a function to generate a % of gaussian noise on some data and return noisyData

```
[ ]: def noisySet(data,noiseLevel):
         guess = np.array([data.x1,data.x2])

         noise = np.random.normal(1,noiseLevel, guess.shape)
         new_signal = guess + noise

         # plt.scatter(new_signal[0],new_signal[1],c=data.labels)
```

```
    new_signal = pd.DataFrame(data={'x1': new_signal[0], 'x2': ␣
↪new_signal[1],'labels':data.labels})
    return new_signal
```

0.4.2 Generate Data for One and Two + Classification

Per dataset, do 5%, 10%, 20%, and 25% noise.

```
[ ]: #Function to generate noisy train,val,test data then run a decision tree + KNN␣
     ↪on these
     def noiseDataModel(train,val,test,noise):



         for types in noise:
             noiseTrain= noisySet(train,types)
             noiseVal = noisySet(val,types)
             noiseTest = noisySet(test,types)

             print("\nNoise Level:", (float(types)*99))
             print("\nDecision Tree")
             decisionTree(noiseTrain,noiseVal,noiseTest)

             print("\nKNN")
             KNNAccuracy(noiseTrain,noiseVal,noiseTest)
```

```
[ ]: #Define the noise levels we want
     noiseLevels = [0.05,0.10,0.20,0.25]

     print("Dataset One")
     noiseDataModel(trainOne,valone,testOne,noiseLevels)
```

```
Dataset One

Noise Level: 4.95

Decision Tree
Validation Accuracy 0.5296296296296297
Test Accuracy 0.546031746031746

KNN
Best K:  1

Validation Accuracy 0.962962962962963
Test Accuracy 0.9714285714285714

Noise Level: 9.9

Decision Tree
```

```
Validation Accuracy 0.5777777777777777
Test Accuracy 0.6174603174603175

KNN
Best K:  1

Validation Accuracy 0.8518518518518519
Test Accuracy 0.8666666666666667

Noise Level: 19.8

Decision Tree
Validation Accuracy 0.5296296296296297
Test Accuracy 0.5444444444444444

KNN
Best K:  5

Validation Accuracy 0.6962962962962963
Test Accuracy 0.6746031746031746

Noise Level: 24.75

Decision Tree
Validation Accuracy 0.4777777777777778
Test Accuracy 0.5190476190476191

KNN
Best K:  9

Validation Accuracy 0.6592592592592592
Test Accuracy 0.6523809523809524
```

[ ]:
```
print("Dataset Two")
factoredUpNoise = [x * 10 for x in noiseLevels]
noiseDataModel(trainTwo,valTwo,testTwo,factoredUpNoise)
```

```
Dataset Two

Noise Level: 49.5

Decision Tree
Validation Accuracy 0.4
Test Accuracy 0.44126984126984126

KNN
Best K:  1
```

```
Validation Accuracy 0.9925925925925926
Test Accuracy 0.9968253968253968

Noise Level: 99.0

Decision Tree
Validation Accuracy 0.3925925925925926
Test Accuracy 0.4380952380952381

KNN
Best K:  1

Validation Accuracy 0.9851851851851852
Test Accuracy 0.9650793650793651

Noise Level: 198.0

Decision Tree
Validation Accuracy 0.3333333333333333
Test Accuracy 0.4158730158730159

KNN
Best K:  1

Validation Accuracy 0.9259259259259259
Test Accuracy 0.8825396825396825

Noise Level: 247.5

Decision Tree
Validation Accuracy 0.3851851851851852
Test Accuracy 0.39365079365079364

KNN
Best K:  1

Validation Accuracy 0.9407407407407408
Test Accuracy 0.892063492063492
```