Assignment Three

I chose to complete programming questions. I will provide the code, and images of my results to demonstrate this in the document here.

DISCLAIMER: The data is generated in a specific way, however if we alter the train data ratio to test data the accuracy of the classifiers will change drastically. As instructed, I try to keep a large portion of data for validation and testing.

Programming with Supervised Learning

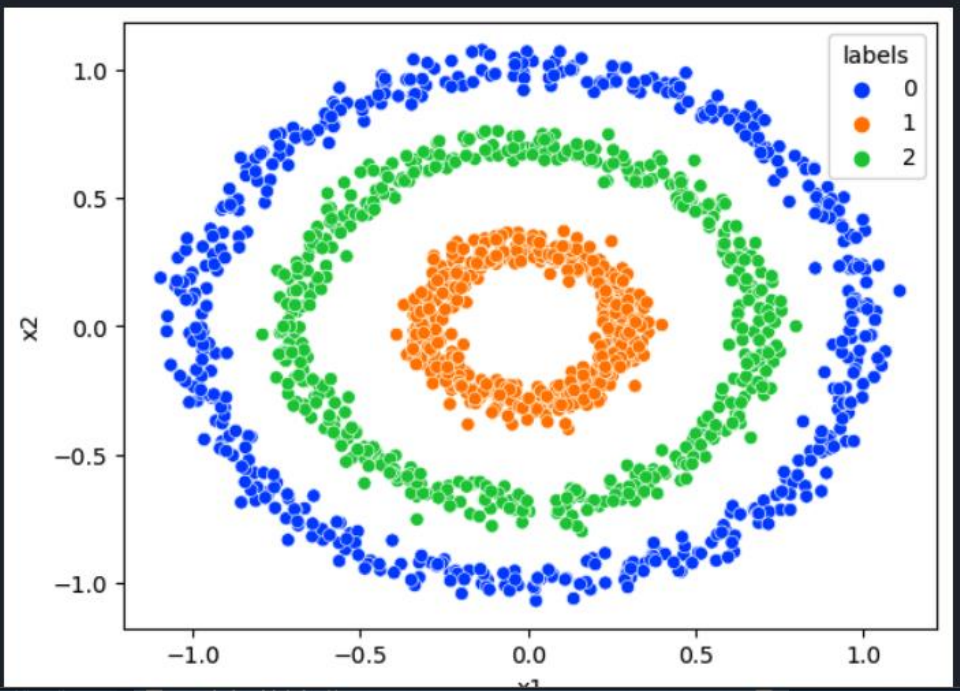
I generated data such as in the images 1, 5 and 6 in the assignment questions.

Generating Data One

```
X_small, y_small = make_circles(n_samples=(SAMPLESIZE,SAMPLESIZE), random_state=3,
noise=0.04, factor = 0.3)
X_large, y_large = make_circles(n_samples=(SAMPLESIZE,SAMPLESIZE), 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])
df.labels.value_counts()
sns.scatterplot(data=df,x='x1',y='x2',hue='labels',palette="bright")
```

[18] ✓ 0.1s

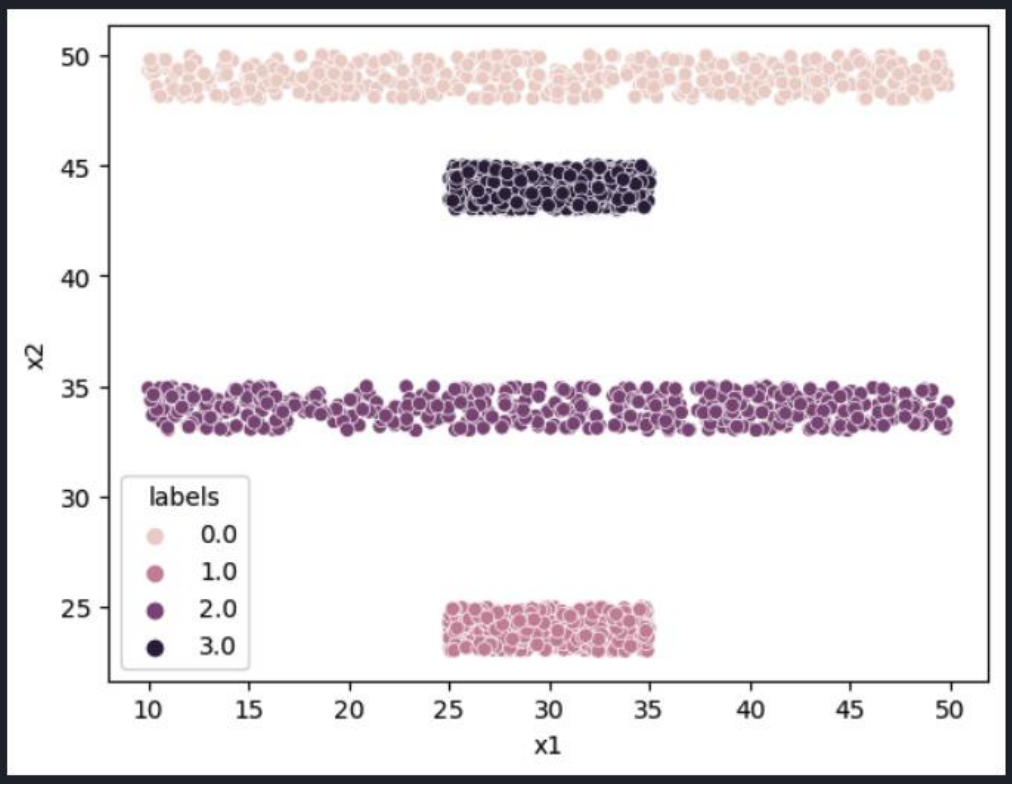
```
<AxesSubplot:xlabel='x1', ylabel='x2'>
```



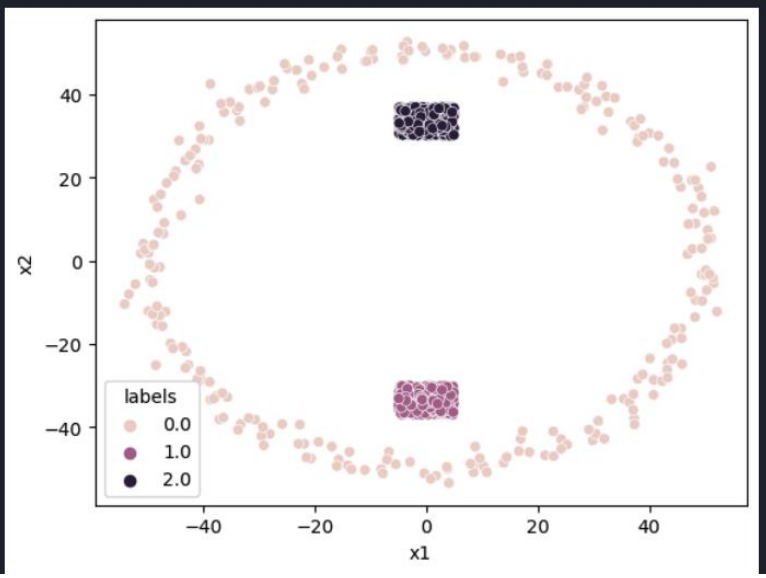
```
    y.extend(list(np.random.uniform(low=23, high=25, size=(SAMPLESIZE,))))
    c.extend(np.ones(SAMPLESIZE))

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

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


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

    sns.scatterplot(data=dfTwo,x='x1',y='x2',hue='labels')
```

[20] ✓ 0.2s

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



```
        y.append(X2[x])

    Xc = list(np.zeros(len(X)))

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

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

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

[22]  ✓ 0.1s

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



I programmed the K-NN algorithm, and then performed cross-validation on for the three pieces of data. I demonstrate the results below:

```
Dataset One
Best K:  3

Validation Accuracy 1.0
Test Accuracy 1.0

Dataset Two
Best K:  1

Validation Accuracy 1.0
Test Accuracy 1.0

Dataset Three
Best K:  3

Validation Accuracy 1.0
Test Accuracy 1.0
```

The accuracy on the data is very high, however the data was generated in a manner where a high accuracy such as this is likely to happen. If we run K-NN with less optimal K values, we achieve less favourable results. My code has the specific implementation in it alongside comments to see how I approached the problem.

Comparison of K-NN and Decision Tree

I used sklearn as a framework to create a decision tree model I could easily run on each dataset. I create the model quite easily with this chosen framework.

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

    #Create decision tree
    clf = DecisionTreeClassifier()
    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))

✓ 0.4s
```

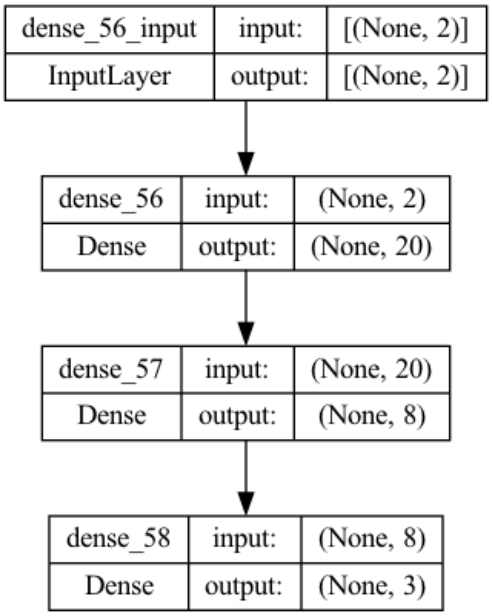
I made a function to make it very simple to run an entire training/cross-validation quickly on a dataset. I ran the function on each dataset and got the following results:

```
Dataset One

Decision Tree
Validation Accuracy 0.45925925925925926
Test Accuracy 0.46984126984126984

KNN
Best K:  1

Validation Accuracy 0.9518518518518518
Test Accuracy 0.9761904761904762

Dataset Two

Decision Tree
Validation Accuracy 0.43703703703703706
Test Accuracy 0.40634920634920635

KNN
Best K:  3

Validation Accuracy 1.0
Test Accuracy 1.0

Dataset Three

Decision Tree
Validation Accuracy 0.5555555555555556
Test Accuracy 0.6285714285714286

KNN
Best K:  3

Validation Accuracy 0.9925925925925926
Test Accuracy 0.9904761904761905
```

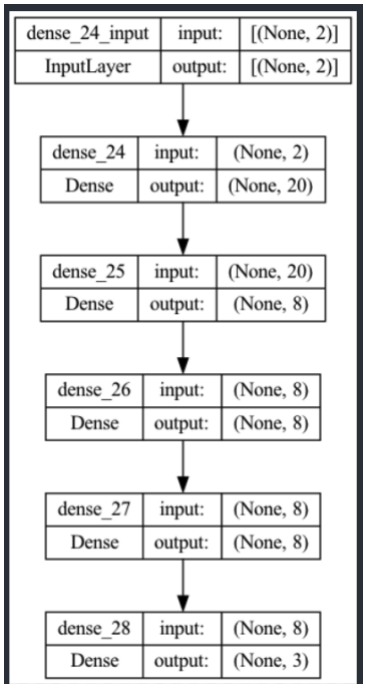
Generally, the KNN algorithm has great classification, whereas the decision tree struggles. The accuracy of the tree on test data never is higher than 63% on any dataset. Changing decision tree parameters and options, plus making a deeper tree may help to increase accuracy here. The specific implementation can be seen in the ipynb file.

Deep Neural Networks

I tested multiple types of networks with different structures in this question. Here we can see the structure of the two networks.



Model One



Model Two

Model two is a deeper model than model one. I constructed more hidden layers for model two, to see how this might affect accuracy. I tested each model on each dataset.

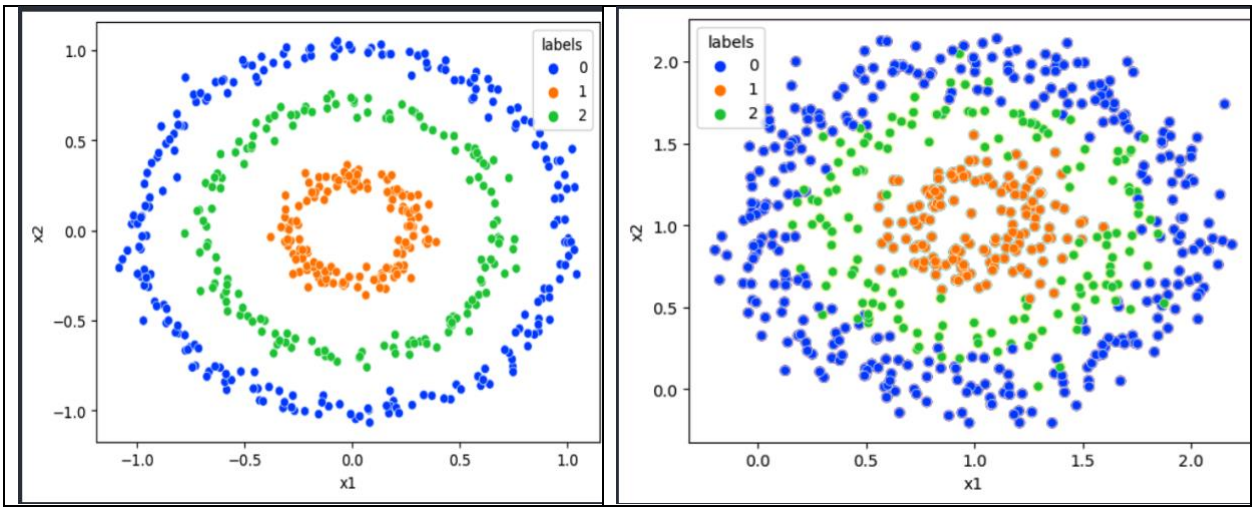
```
runModelOne(trainOne,valone, testOne)
✓ 1.6s

2022-12-06 19:05:05.336653: W tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU frequency: 0 Hz

9/9 [==============================] - 0s 738us/step - loss: 0.3731 - accuracy: 0.9148
Validation Accuracy: 91.48
20/20 [==============================] - 0s 592us/step - loss: 0.4045 - accuracy: 0.8873
Test Accuracy: 88.73


runModelOne(trainTwo,valTwo,testTwo)
✓ 0.9s

5/5 [==============================] - 0s 856us/step - loss: 0.9653 - accuracy: 0.6000
Validation Accuracy: 60.00
10/10 [==============================] - 0s 746us/step - loss: 0.9868 - accuracy: 0.5492
Test Accuracy: 54.92


runModelOne(trainThree,valThree,testThree)
✓ 0.9s

5/5 [==============================] - 0s 913us/step - loss: 0.3336 - accuracy: 0.9407
Validation Accuracy: 94.07
10/10 [==============================] - 0s 732us/step - loss: 0.2228 - accuracy: 0.9587
Test Accuracy: 95.87
```

```
runModelTwo(trainOne,valone, testOne)
[22] ✓ 1.9s
...  9/9 [==============================] - 0s 776us/step - loss: 0.0503 - accuracy: 0.9926
     Validation Accuracy: 99.26
     20/20 [==============================] - 0s 594us/step - loss: 0.0472 - accuracy: 0.9968
     Test Accuracy: 99.68

runModelTwo(trainTwo,valTwo,testTwo)
[23] ✓ 1.9s
...  5/5 [==============================] - 0s 1ms/step - loss: 1.2817 - accuracy: 0.4815
     Validation Accuracy: 48.15
     10/10 [==============================] - 0s 839us/step - loss: 1.3054 - accuracy: 0.4571
     Test Accuracy: 45.71

runModelTwo(trainThree,valThree,testThree)
[24] ✓ 1.4s
...  5/5 [==============================] - 0s 1ms/step - loss: 0.3305 - accuracy: 0.9333
     Validation Accuracy: 93.33
     10/10 [==============================] - 0s 1ms/step - loss: 0.1985 - accuracy: 0.9587
     Test Accuracy: 95.87
```

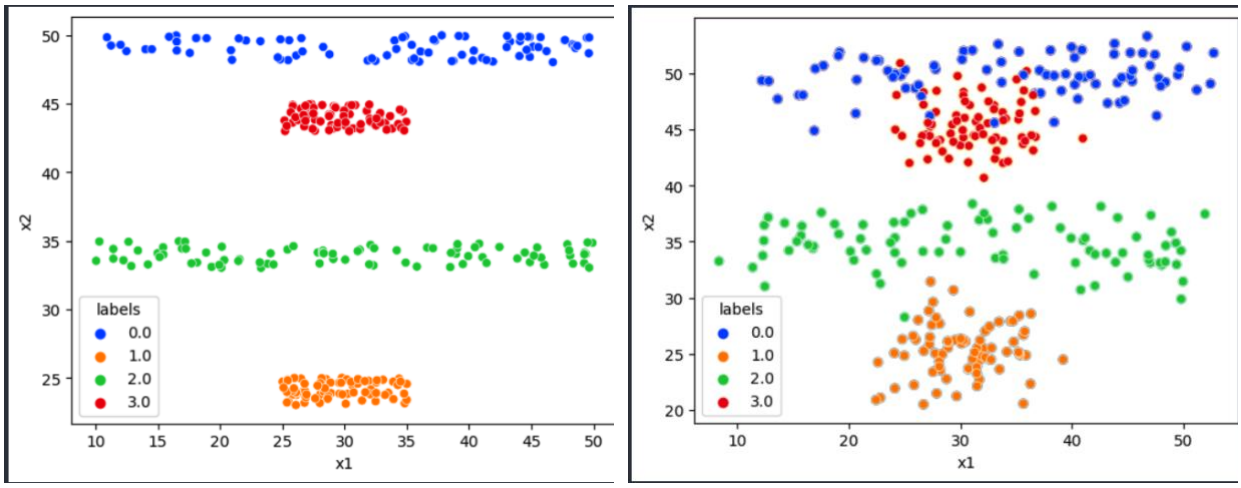
Model two has better accuracy generally for test and validation data. Dataset two isn't well suited to the neural network constructed, and we can see even lower accuracy with the deeper model here. Specific implementation is available in the ipynb file. Based on the testing results, the model to use is data dependant. Model two should be used on datasets one, and three, whereas model one should be used for dataset two. This is based on a visual inspection of the test accuracy.

Noisy Data

I added a variety of noise to the different data. I chose datasets one and two to generate noise with. Below I show some noise added to each.



Here above I added 10% noise to the data on the right. The data on the left is without noise to demonstrate the effects. Additionally, dataset two was also modified.



Similarly, the left dataset demonstrates the data with no noise, where the right dataset involves noise. I made a list of noise values to test, and then iterated through each dataset applying noise. In each iteration I ran the K-NN and Decision tree functions I defined earlier which gave me validation and test accuracy. Below we can see the output of this.

```
Dataset One

Noise Level: 4.95

Decision Tree
Validation Accuracy 0.5037037037037037
Test Accuracy 0.4857142857142857

KNN
Best K:  1

Validation Accuracy 0.9777777777777777
Test Accuracy 0.9619047619047619

Noise Level: 9.9

Decision Tree
Validation Accuracy 0.5037037037037037
Test Accuracy 0.4857142857142857

KNN
Best K:  5

Validation Accuracy 0.8407407407407408
Test Accuracy 0.8285714285714285

Noise Level: 19.8

Decision Tree
Validation Accuracy 0.5037037037037037
Test Accuracy 0.4857142857142857

KNN
Best K:  9

Validation Accuracy 0.7148148148148148
Test Accuracy 0.7238095238095238

Noise Level: 24.75

Decision Tree
Validation Accuracy 0.5037037037037037
Test Accuracy 0.4857142857142857

KNN
Best K:  9

Validation Accuracy 0.6703703703703704
```

```
Dataset Two

Noise Level: 49.5

Decision Tree
Validation Accuracy 0.4074074074074074
Test Accuracy 0.37142857142857144

KNN
Best K:  1

Validation Accuracy 0.9925925925925926
Test Accuracy 0.9936507936507937

Noise Level: 99.0

Decision Tree
Validation Accuracy 0.4444444444444444
Test Accuracy 0.4095238095238095

KNN
Best K:  1

Validation Accuracy 0.9777777777777777
Test Accuracy 0.9873015873015873

Noise Level: 198.0

Decision Tree
Validation Accuracy 0.3851851851851852
Test Accuracy 0.3873015873015873

KNN
Best K:  5

Validation Accuracy 0.8666666666666667
Test Accuracy 0.9015873015873016

Noise Level: 247.5

Decision Tree
Validation Accuracy 0.4
Test Accuracy 0.37777777777777777

KNN
Best K:  3

Validation Accuracy 0.8666666666666667
```

The addition of noise to each of the datasets make classification more difficult. The KNN algorithm seems to find classification much more difficult as we increase noise in each case, but the decision tree accuracy doesn't decrease too much. The KNN algorithm additionally keeps changing to a different optimal K value. The noise can help to determine good hyperparameters for the KNN algorthim. For dataset one, 9 is likely the best candidate as even with 25% noise, an accuracy of 67% is achieved. For dataset two, 3 is certainly a good candidate as even with 25%(247.5 is the scaled value with the data) noise, we achieve an accuracy of 86% on unseen data despite noisiness in training data.