k-Nearest Neighbour

1.
Class labels:
```
[3, 3, 3, 1, 1, 1, 1, 1, 1, 2, 2, 3, 3, 3, 1, 2, 3, 3, 1, 1, 3, 2, 2, 3, 2, 3,
2, 3, 2, 1, 2, 1, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2, 3, 1, 2, 1, 3, 2, 2, 1, 3, 1,
1, 3, 3, 1, 1, 3, 1, 3, 3, 2, 2, 3, 2, 3, 3, 1, 1, 2, 1, 3, 2, 2, 1, 1, 1, 3,
1, 1, 2, 2, 3, 1, 2, 1, 1, 2, 1]
```
K=1 accuracy: 0.9438202247191011

2.
K=3 accuracy: 0.9213483146067416
We can see here that k=3 scored lower on the test set compared to k=1. This is likely due to some outliers in the data training data. When the 3 lowest distances are being chosen some of those distance may be outliers in the data which is then taken into consideration as closest neighbours.

3.
Advantages:

- There is no training period, it simply stores the dataset and then compares the test data with it. This also means more training data can be added without re-training.
- It is very simple and easy to implement.

Disadvantages:

- It is sensitive to noisy data, for example, an outlier could be the nearest neighbor which would mean the outcome is wrong.
- As the datasets become larger the time taken to calculate the distances takes longer.
- Feature scaling needs to be applied to make the outcomes accurate.

4.
Accuracy = []
For I in range 5
  Split the data into training and test where i is the portion of the data
  Remove the labels from the test data
  Normalize the data
  Use the train data to predict the test labels
  Check the accuracy of the predictions and add them to the accuracy lists
Take the average of all the 5 accuracies

5.
K-means clustering

- Initialize 3 random means from the data set as we have 3 classes in our dataset
- Create 3 clusters by assigning each instance to the nearest cluster using the nearest mean calculation

- Take the mean of each cluster and set them as the new 3 means
- Repeat each step until convergence

Decision Tree Learning

1.

Accuraccies

```
Training accuracy: 1.0 | 112/112
Test accuracy: 0.76 | 19/25
```

Full tree

```
ASCITES = True
   SPIDERS = True
      VARICES = True
         STEROID = True
            Class live prob = 1.0
         STEROID = False
            SPLEENPALPABLE = True
               FIRMLIVER = True
                  Class live prob = 1.0
               FIRMLIVER = False
                  BIGLIVER = True
                     SGOT = True
                        Class live prob = 1.0
                     SGOT = False
                        FEMALE = True
                           Class live prob = 1.0
                        FEMALE = False
                           ANOREXIA = True
                              Class die prob = 1.0
                           ANOREXIA = False
                              Class live prob = 1.0
                  BIGLIVER = False
                     Class live prob = 1.0
            SPLEENPALPABLE = False
               ANOREXIA = True
                  Class live prob = 1.0
               ANOREXIA = False
                  Class die prob = 1.0
      VARICES = False
         Class die prob = 1.0
   SPIDERS = False
      FIRMLIVER = True
         ANOREXIA = True
            SGOT = True
               Class live prob = 1.0
            SGOT = False
               Class die prob = 1.0
         ANOREXIA = False
            Class live prob = 1.0
      FIRMLIVER = False
         SGOT = True
            BIGLIVER = True
               Class live prob = 1.0
            BIGLIVER = False
               Class die prob = 1.0
         SGOT = False
            Class live prob = 1.0
```

```
ASCITES = False
   BIGLIVER = True
      VARICES = True
         FIRMLIVER = True
            STEROID = True
               Class die prob = 1.0
            STEROID = False
               BILIRUBIN = True
                  Class live prob = 1.0
               BILIRUBIN = False
                  Class die prob = 1.0
         FIRMLIVER = False
            Class live prob = 1.0
      VARICES = False
         Class die prob = 1.0
   BIGLIVER = False
      Class live prob = 1.0
```

My decision tree on the training data classified the training data with 100% accuracy, this should be the case as it is based of the training data so every possible path down the tree is there. Compared to the baseline classifier on the training data of 0.8125 (81.25%) the decision tree did much better.
The baseline classifier for the testing data was 0.8 (80%) compared to the test data on the decision tree which classified with an accuracy of 0.76 (76%). This is lower but it does take in account every feature where as the baseline only looks as the class. Thus if the class is not part of the 80% class then there will only be a 20% accuracy. We also are not taking into account the fact that using more training data will help improve the Deccison tree but the baseline classifier will not become more accurate it will still be based of the majority class.

2.

```
Kfold= 0 Accuracy: 0.7666666666666667
Kfold= 1 Accuracy: 0.8
Kfold= 2 Accuracy: 0.8
Kfold= 3 Accuracy: 0.8
Kfold= 4 Accuracy: 0.7666666666666667
Kfold= 5 Accuracy: 0.7666666666666667
Kfold= 6 Accuracy: 0.8666666666666667
Kfold= 7 Accuracy: 0.2
Kfold= 8 Accuracy: 0.7
Kfold= 9 Accuracy: 0.8
Average accuracy accros 10 folds: 0.7266666666666667
```

From running the 10 folds we can see that fold 7 is very low. Using the mean of all 10 folds we can see the average is still 7.26 (72.6%)
(See python doc for implementation)

3.
(a)To prune the decision tree using one of the common methods, we can add a max depth value that stops the recursiona at certain depth. (b)The reason it reduces the accuracy on the training set is because it doesnt build the tree all the way down so when it reaches a leaf node during training it will have to figure out what class the instance belongs to. (c)The reason it might

however improve on the testing set is due to overfitting. If you do not prune the tree overfitting can occur making the tree to specific to the training data. Thus if you prune there is less likely to be overfitting and the test data will be classified with more accuracy.

4.
The reason the impurity measure is not appropriate for more than two classes is because as soon as one impurity is 0 or a small value the result of the whole calculation will be 0 or a really small number making it very hard to distinguish between different results among features during comaprisions

Perceptron

1.
Using the random weights with a constant seed of 0 it took my perceptron 380 epochs to learn the weights to achieve 90% accuracy. If I used random weights the starting accuracy was sometimes higher but it would still converge around 80% to 90%

2.
Because the data you are using to train the weights is specific to the test labels which means it will fit better. To test it properly the data needs to be split into training and testing. The testing data can then be used to gain a better understanding of the actual accuracy of the perceptron. (See part3.py for an example)