# Daniel Walker - 300336857

## Part 1 - KNN

### Question 1

KNN Success rate: 92.0% accuracy
69/75 With a K value = 1

```
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-setosa ==> Predicted instance: Iris-setosa
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-virginica
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-versicolor ==> Predicted instance: Iris-versicolor
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-versicolor
```

```
Given instance: Iris-virginica ==> Predicted instance: Iris-versicolor
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-versicolor
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-versicolor
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-versicolor
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
Given instance: Iris-virginica ==> Predicted instance: Iris-virginica
```

## Question 2

Report the classification accuracy on the test set of the k-nearest neighbour method where k=3, and compare and comment on the performance of the two classifiers (k=1 and k=3);

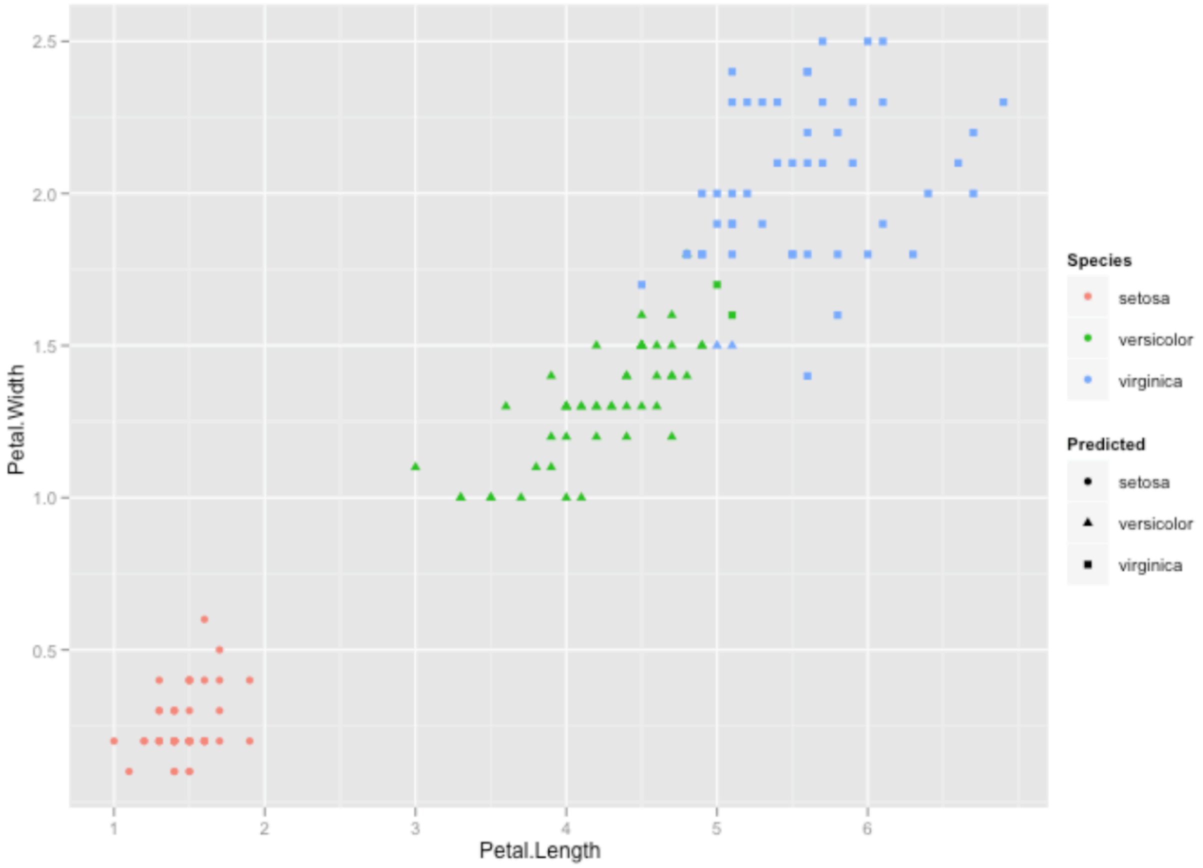
Both K=1 and K=3 have the same accuracy, with 69 out of 75 were predicted correctly.

Typically, a larger K will generate more accurate predictions. If we assigned K to equal the size of the training data, then, the most common Iris would be assigned the “class” for all these iris that were classified, which is a very inaccurate prediction. Whereas if we only had K=1, we would only be looking at the next closes iris, which is not representative of all the data clusters.

As stated above, k=1 and k=3 are giving the same accuracy, this most likely is due to the three flower types are clustered. Meaning, for most cases, the next closes iris is one from the same cluster - leading to a correct prediction.
As shown on the above diagram, the iris-Setoa species will not be affected if K is 1 or 3, as the closes iris is always going to be another setoa flower. versi-colour and virginica have a little overlap, so there will be a few mis-predictions, as shown in my accuracy being only 92%

| KNN Results | k=1 | k=3 | k=4 | k=5 |
|---|---|---|---|---|
| Success rate: | 92% | 92% | 93.33% | 92% |

```
run_01: Success rate: 86.48648648648648 (32/37)
run_02: Success rate: 91.8918918918919 (34/37)
run_03: Success rate: 86.48648648648648 (32/37)
run_04: Success rate: 81.08108108108108 (30/37)
run_05: Success rate: 81.08108108108108 (30/37)
run_06: Success rate: 83.78378378378379 (31/37)
run_07: Success rate: 86.48648648648648 (32/37)
run_08: Success rate: 83.78378378378379 (31/37)
run_09: Success rate: 62.16216216216216 (23/37)
run_10: Success rate: 86.48648648648648 (32/37)
```



# Question 3

Discuss the main advantages and disadvantages of k-Nearest Neighbour method.

| Advantages | Disadvantages |
|---|---|
| **Robust to noisy data** | Need to manually determine K nearest neighbours |
| **Effective if training data set is large** | Computation cost is high as need to calculate distances for every training value |
| **Simple to implement** | Must know we have the correct distance parameter/don't know what the best parameter should be to calculate Euclidean distance |
| | |

# Question 4

Assuming that you are asked to apply the k-fold cross validation method for the above problem with k=5, what would you do? State the major steps.

1. Partition data into 5-folds (k-folds).
2. For each of the 5 sub-sets, we want to treat one as the test set, using the remainder k-1 subsets as the training sets.
3. Repeat K times for each subset, treating each subset as a testing set once, while the rest are a training set.

4. Average the results to produce one single estimation of how the algorithm will perform on real data.

## Question 5

In the above problem, assuming that the class labels are not available in the training set and the test set, and that there are three clusters, which method would you use to group the examples in the data set? State the major steps.

It would make sense to use k-means clustering as it can classify unknown entries that don't have class labels into clusters. Using k=3 for this iris data, it would group the irises into three clusters, each having their own class label.

1. Make k centroids
2. Randomly place the centroids within the range of the data set
3. For each centroid, loop over the assigned centroids and calculate the mean for each attribute.
4. Set the mean values to be the property of the centroid, so it becomes the centre of the cluster.
5. Repeat until no change

# Part 2 - Decision Tree

## Question 1

Success rate: 86.9% (119/137)
BaseLine: Success rate: 81% (111/137) (this was run by manually calling a different method, it doesn't run simultaneously with the regular tree.

Decision Tree:

```
 ASCITES = True:
  SPIDERS = True:
   VARICES = True:
    HISTOLOGY = True:
     MALAISE = True:
      FATIGUE = True:
       BILIRUBIN = True:
        FEMALE = True:
         Class dieProb: 1.0
        FEMALE = False:
         SPLEENPALPABLE = True:
          ANOREXIA = True:
           SGOT = True:
            ANTIVIRALS = True:
             FIRMLIVER = True:
              AGE = True:
               BIGLIVER = True:
                STEROID = True:
                 Class dieProb: 0.8113207547169812
                STEROID = False:
                 Class liveProb: 0.8085106382978723
               BIGLIVER = False:
                Class liveProb: 0.8333333333333334
              AGE = False:
               Class liveProb: 0.8181818181818182
             FIRMLIVER = False:
              Class liveProb: 0.7857142857142857
            ANTIVIRALS = False:
```

```
            Class liveProb: 0.875
          SGOT = False:
           Class liveProb: 0.8311688311688312
         ANOREXIA = False:
          Class liveProb: 0.7272727272727273
        SPLEENPALPABLE = False:
         Class liveProb: 0.7142857142857143
      BILIRUBIN = False:
       Class liveProb: 0.926829268292683
     FATIGUE = False:
      Class liveProb: 0.7352941176470589
    MALAISE = False:
     Class liveProb: 0.65
   HISTOLOGY = False:
    Class liveProb: 0.9285714285714286
  VARICES = False:
   Class dieProb: 0.6153846153846154
 SPIDERS = False:
  Class liveProb: 0.5483870967741935
ASCITES = False:
 Class dieProb: 0.6666666666666666
```

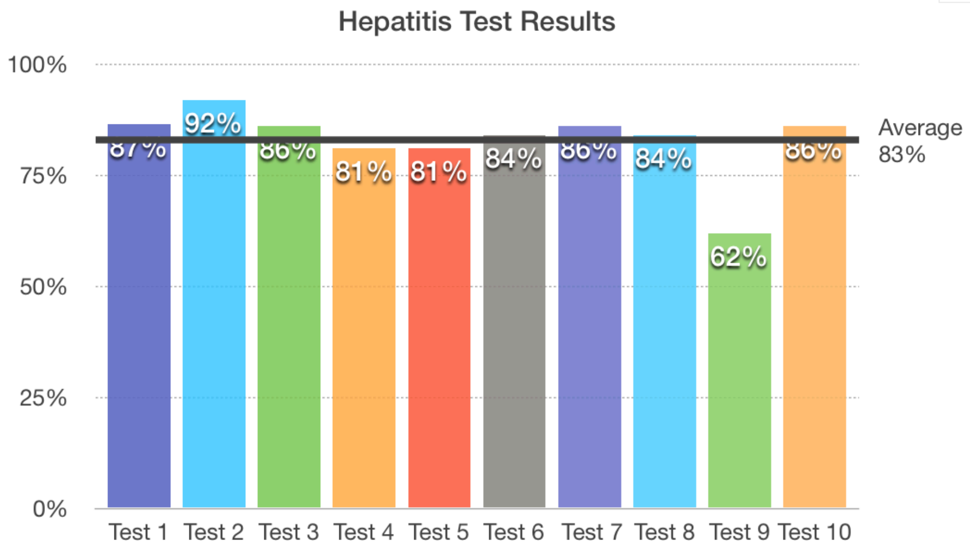
I believe my DT implementation is slightly wrong. On Friday 13th April I discovered my implementation does not split the instances as the tree is built. I did not have time to fix this error unfortunately

The “baseline probability”, is the probability of the most occurring class (outcome) occurring. In my case, 111 live out of 137 people (0.81 probability of living). Hence why our success rate when running the program on the BaseLine we get this accuracy of prediction. This is providing less insight to the algorithm than my fully implemented method is, hence my success rate is 86.9% - improving the prediction accuracy over the baseline classifier by 5.9%

To improve the accuracy of this, I would like to use a bigger testData set as only proving the Decision tree algorithm with 137 lines of data is not large enough to represent the disease correctly and for it to provide 100% accurate predictions.

## Question 2

run_01: Success rate: 86.5% (32/37)
run_02: Success rate: 91.9% (34/37)
run_03: Success rate: 86.5% (32/37)
run_04: Success rate: 81.1% (30/37)
run_05: Success rate: 81.1% (30/37)
run_06: Success rate: 83.8% (31/37)
run_07: Success rate: 86.5% (32/37)
run_08: Success rate: 83.8% (31/37)
run_09: Success rate: 62.2% (23/37)
run_10: Success rate: 86.5%(32/37)



Average = 307/370 = 82.9%

# Question 3

a. **Pruning leaves from the DT:**

Improving accuracy can be achieved on this algorithm by pruning leaves from the tree in the testData. We do this by splitting a classified data set into training and testing data. Next we then **learn** on the **training** data as we normally would, and run the testing data measuring and recording the performance. Now we begin the pruning. For each node in the tree we want to pretend that node (and its children) don't exist and we measure the performance using the testing data. We then remove the node that results in the greatest improvement and repeat the above steps until further pruning is harmful.

b. **Why it would reduce accuracy on the training set:**

While learning, the DT algorithm will keep splitting the data until it **ends** with **pure** sets. This is bad as it will split until the leaves have just one example each (very specific properties). This means for it to accurately work in the future, it has something we have already seen in the past. This is called overfitting. When learning the training set, the algorithms accuracy will reach 100% if the training data was to be tested on itself. This is because we have learnt every case and have an exact match for each instance, so we can accurately predict the outcome. When we start pruning the the tree, we are **taking these exact matches away** so it is going to be **less accurate** when running the algorithm on the training set. This may seem like a bad thing, but the whole idea of the algorithm is to improve the accuracy when testing new, unclassified data.

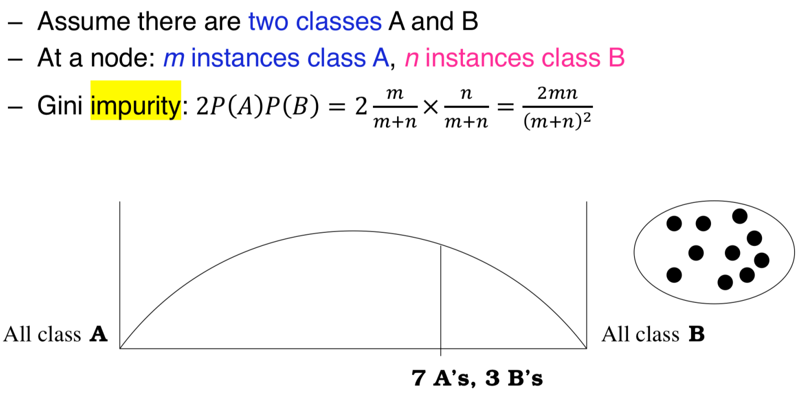
c. **Why it might improve accuracy on the test set.**

When running a new, unseen test set on the pruned tree, we **will see improvements**. This is because we have **reduced the 'overfitting'** for specific case, and now have a more generalised tree, which is better suited for unknown data.

# Question 4

Explain why the impurity measure is not a good measure if there are three or more classes that the decision tree must distinguish.

Impurity is calculated between two classes (Class **A** and **B**, as shown in the diagram to the right). If we were to expand the formula out and allow it to to calculate the purity for three nodes then the formula would look like this.



- Assume there are two classes A and B
- At a node: $m$ instances class A, $n$ instances class B
- Gini impurity: $2P(A)P(B) = 2\frac{m}{m+n} \times \frac{n}{m+n} = \frac{2mn}{(m+n)^2}$

P(A) P(B) P(C) = (a / (a+b+c) ) * (b / (a+b+c) ) * (c / (a+b+c) )

If we have one of the purities being zero, the whole purity will be zero.

# Part 3 Perceptron

## Question 1

Part 3 code not attempted

## Question 2

2. Explain why evaluating the perceptron’s performance on the training data is not a good measure of its effectiveness. You may wish to create additional data to get a better measure. If you do, report on the perceptron’s performance on this additional data.

Its a bad idea to measure the Perceptrons performance on the training data because it has already learnt it. A better way to measure its performance would be to using test data (data which it hasn’t seen yet).