COMP307

Assignment 1

Basic Machine Learning Algorithms

William Kilty

300473541

Part 1:

1. The output of k = 1 is as shown to the right:

   Where accuracy for K = 1 is 94.38202247191012%

2. The accuracy when K = 3 is 95.50561797752809%

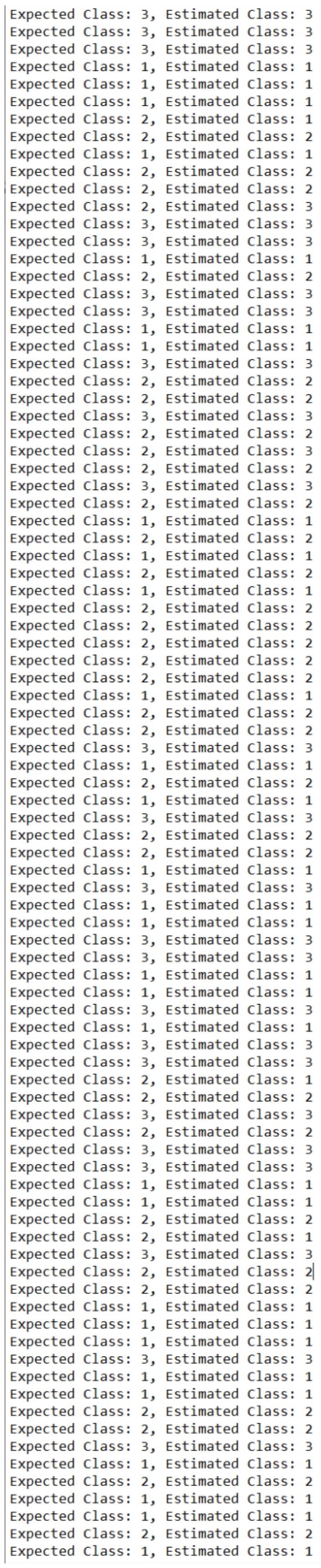
   K=3 is 1.2% better performing in this case than K=1. This is an improvement, but not by much. This could indicate that these are well defined groups with obvious distinctions between them, and without many outliers. With more overlap of data, we would expect a much larger difference than we have observed.

3. Advantages of this method include its speed - it doesn't require any training time beyond normalising the training data - and its lack of 'overfitting' - with a higher K value, the system will ignore outliers, providing a robust predictive model.

   Disadvantages are that if two possibilities are 'just as likely' the model can only guess randomly. If K is low, it will pick up outliers and make inaccurate predictions. It only looks at distance, meaning if there is a non-linear trend, the model has no way of picking up on it. Plus, it cannot adapt at all. I wouldn't call it Machine Learning.

4. Divide the dataset into five equal groups; select one as the test data; use the other four as training data; take results; repeat with each of the five groups acting as the test data; combine each of the results into an average score, showing more accurately how our method performs on the data
5. K-Means Clustering:

   Select K random points from the dataset; attach each point to its nearest of the k 'means'; replace each mean with the mean of this new cluster; repeat until we have k distinct clusters.



```
Expected Class: 3, Estimated Class: 3
Expected Class: 3, Estimated Class: 3
Expected Class: 3, Estimated Class: 3
Expected Class: 1, Estimated Class: 1
Expected Class: 1, Estimated Class: 1
Expected Class: 1, Estimated Class: 1
Expected Class: 2, Estimated Class: 1
Expected Class: 2, Estimated Class: 2
Expected Class: 1, Estimated Class: 1
Expected Class: 2, Estimated Class: 2
Expected Class: 2, Estimated Class: 2
Expected Class: 2, Estimated Class: 3
Expected Class: 3, Estimated Class: 3
Expected Class: 3, Estimated Class: 3
Expected Class: 1, Estimated Class: 1
Expected Class: 2, Estimated Class: 2
Expected Class: 3, Estimated Class: 3
Expected Class: 3, Estimated Class: 3
Expected Class: 1, Estimated Class: 1
Expected Class: 1, Estimated Class: 1
Expected Class: 3, Estimated Class: 3
Expected Class: 2, Estimated Class: 2
Expected Class: 2, Estimated Class: 2
Expected Class: 3, Estimated Class: 3
Expected Class: 2, Estimated Class: 2
Expected Class: 2, Estimated Class: 3
Expected Class: 2, Estimated Class: 2
Expected Class: 3, Estimated Class: 3
Expected Class: 2, Estimated Class: 2
Expected Class: 1, Estimated Class: 1
Expected Class: 2, Estimated Class: 2
Expected Class: 1, Estimated Class: 1
Expected Class: 2, Estimated Class: 2
Expected Class: 1, Estimated Class: 1
Expected Class: 2, Estimated Class: 2
Expected Class: 2, Estimated Class: 2
Expected Class: 2, Estimated Class: 2
Expected Class: 2, Estimated Class: 2
Expected Class: 2, Estimated Class: 2
Expected Class: 1, Estimated Class: 1
Expected Class: 2, Estimated Class: 2
Expected Class: 2, Estimated Class: 2
Expected Class: 3, Estimated Class: 3
Expected Class: 1, Estimated Class: 1
Expected Class: 2, Estimated Class: 2
Expected Class: 1, Estimated Class: 1
Expected Class: 3, Estimated Class: 3
Expected Class: 2, Estimated Class: 2
Expected Class: 2, Estimated Class: 2
Expected Class: 1, Estimated Class: 1
Expected Class: 3, Estimated Class: 3
Expected Class: 1, Estimated Class: 1
Expected Class: 1, Estimated Class: 1
Expected Class: 3, Estimated Class: 3
Expected Class: 3, Estimated Class: 3
Expected Class: 1, Estimated Class: 1
Expected Class: 1, Estimated Class: 1
Expected Class: 3, Estimated Class: 3
Expected Class: 1, Estimated Class: 1
Expected Class: 3, Estimated Class: 3
Expected Class: 3, Estimated Class: 3
Expected Class: 2, Estimated Class: 1
Expected Class: 2, Estimated Class: 2
Expected Class: 3, Estimated Class: 3
Expected Class: 2, Estimated Class: 2
Expected Class: 3, Estimated Class: 3
Expected Class: 3, Estimated Class: 3
Expected Class: 1, Estimated Class: 1
Expected Class: 1, Estimated Class: 1
Expected Class: 2, Estimated Class: 2
Expected Class: 2, Estimated Class: 1
Expected Class: 3, Estimated Class: 3
Expected Class: 2, Estimated Class: 2
Expected Class: 2, Estimated Class: 2
Expected Class: 1, Estimated Class: 1
Expected Class: 1, Estimated Class: 1
Expected Class: 1, Estimated Class: 1
Expected Class: 3, Estimated Class: 3
Expected Class: 1, Estimated Class: 1
Expected Class: 1, Estimated Class: 1
Expected Class: 2, Estimated Class: 2
Expected Class: 2, Estimated Class: 2
Expected Class: 3, Estimated Class: 3
Expected Class: 1, Estimated Class: 1
Expected Class: 2, Estimated Class: 2
Expected Class: 1, Estimated Class: 1
Expected Class: 1, Estimated Class: 1
Expected Class: 2, Estimated Class: 2
Expected Class: 1, Estimated Class: 1
```

Part 2:

1. The output is as follows:

```
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
```

```
Testing:
~~~~~~~~~~

Class die, prob = 1.0, Actual: { live true true false true false true true true true false false true false true false true }
Class live, prob = 1.0, Actual: { die false false false true false false false true true false false true false true false true }
Class live, prob = 1.0, Actual: { live false false false true false false false true true true false false true true true true }
Class live, prob = 1.0, Actual: { live false false true true false false true true true true true true true false true false }
Class live, prob = 1.0, Actual: { live false false false true false false false false false true false true false true false true }
Class live, prob = 1.0, Actual: { live false false false true true true true true true true true true true false false false }
Class live, prob = 1.0, Actual: { die false false true true false true true true false false false true false true true true }
Class live, prob = 1.0, Actual: { live false false true false true true true true true true true true true false false false }
Class live, prob = 1.0, Actual: { live false false true true true true true true true true true true true false false true }
Class live, prob = 1.0, Actual: { live false false false true false true true true false false true true true true false false }
Class live, prob = 1.0, Actual: { live false false true false true true true true true true true true true true false true }
Class live, prob = 1.0, Actual: { live false false true true true true true true false true false true false true true true }
Class live, prob = 1.0, Actual: { live false false true true false true true true true true true true true false false false }
Class live, prob = 1.0, Actual: { live false false false true false false true true true true true true true true false false }
Class live, prob = 1.0, Actual: { die true false false true false false true true true true false false true true false true }
Class live, prob = 1.0, Actual: { die true false false true false false true true false false false true true true true true }
Class live, prob = 1.0, Actual: { live false false false true true true true true true false true true true false false false }
Class live, prob = 1.0, Actual: { live false false false false false false false false false true true true true false false true }
Class die, prob = 1.0, Actual: { die false false false false false false true true true true false true true false false true }
Class live, prob = 1.0, Actual: { live true false true true false true true true true true true true true true false false }
Class live, prob = 1.0, Actual: { live true false false false true true true true true true true true true false false false }
Class die, prob = 1.0, Actual: { live true true false true false true true false false false false true true false true true }
Class live, prob = 1.0, Actual: { live true false true true true true true true true true true true true true false true }
Class live, prob = 1.0, Actual: { live false false true true true true true true true true true true true true true false }
Class live, prob = 1.0, Actual: { live true false false true false true true true true true true true true false false false }

DT Tests Accuracy: 76.00%
Baseline Accuracy: 80.00%

DT Run 0 Accuracy: 86.67%
DT Run 1 Accuracy: 80.00%
DT Run 2 Accuracy: 66.67%
DT Run 3 Accuracy: 70.00%
DT Run 4 Accuracy: 80.00%
DT Run 5 Accuracy: 66.67%
DT Run 6 Accuracy: 80.00%
DT Run 7 Accuracy: 76.67%
DT Run 8 Accuracy: 66.67%
DT Run 9 Accuracy: 73.33%

Average accuracy from 10-fold cross-validation: 74.67%
```

Accuracy of the decision tree is 19/25, or 76%.
The Baseline Predictor produces 20/25, or 80%.

This is generally not ideal, as we'd like our model to be better than just assuming everyone will live. This is likely due to overfitting. Our generated tree is such that the test suite is 100% accurate. So anything that doesn't match 100% with the test suite is likely going to be wildly inaccurate. The model isn't finding patterns that indicate survivability, it's making the tests pass.

2. From the output above, you can see the accuracy rating of each of the 10 folds. To calculate the average, we sum each of the 10 results, and then divide the total by 10.

   $$\frac{86.67+80+66.67+70+80+66.67+80+76.67+66.67+73.33}{10}$$

   This gives us an average accuracy of 74.67. This is Even worse than our original test. This seriously indicates that our model is victim to overfitting.
3.
   a. One way to prune leaves would be to give the decision-tree a maximum depth. Once we have looked at a certain number of attributes, we create a leaf node and just accept the impurity we're left with. Another method would be to look for a certain impurity threshold, that once we get below, we stop looking for attributes and create a leaf.
   b. This will reduce the accuracy of the training set as we're leaving impurities in our leaf nodes. So we can't be 100% sure of an outcome in the test data. This means we have to start guessing at some points, reducing our training score.
   c. However, this is likely to improve our score in the test sets, as now we aren't overfitting as I've discussed above. We're looking at the general trends, and taking an educated guess for very specific cases. This is likely to produce a much better score.
4. When we take the impurity for two categories, we can assume that one category missing means we've only got one category present, giving a pure set. For example, say we have a set of one element.

   $$\boxed{\quad O \quad}$$

   Our impurity measurement is as follows: $\frac{1}{1} * \frac{0}{1}$ giving an impurity of 0.

   However, say we have three categories, and an example such as this.

   $$\boxed{\quad O \quad X \quad}$$

   Our impurity measurement is as follows: $\frac{1}{2} * \frac{1}{2} * \frac{0}{2}$ giving an impurity of 0.

   But we know this set is not pure. This is an issue, and is why we cannot use this method for more than two categories

Part 3:

1. My perceptron managed an accuracy of 95.16%. It was unable to reach an accuracy of 100%. I suspect this is due to the data not being linearly separable. So with a single perceptron, 95% is about the upper limit of what is possible. The accuracy would fluctuate a fair amount to start off with, but level off and slowly creep up after that, to around 95%.
   The output of the program was as follows:
   ```
   It took 698 cycles through the dataset,
   with 244762 total training calls on the Perceptron
   to reach an accuracy of 95.16%.

   This leaves 17 of the 351 total instances estimated incorrectly

   The weights the Perceptron used to reach this accuracy were
   (-17.59999999999997, 14.999999999999979, 0.0, 1.3308820000001, -0.427944000000019,
   2.248384000000047, 2.042931999999741, 0.2860739999999686, 2.959788000000039, 3.285031999999522,
   0.8461780000000952, -2.465732000000381, -0.7294040000000526, -0.6800780000000184,
   -0.7267439999999841, 2.8104360000003648, -2.0262359999997996, 0.8639579999999472,
   1.1153680000000004, -3.4674600000001594, 0.49064199999990765, 0.028435999999934305,
   -2.4266720000004325, 1.7767979999999306, 1.2796159999998962, 0.7989640000000404,
   0.017814000000076925, -2.730302000000082, 0.08509799999998878, 1.581722000000101,
   1.6413239999998517, 0.4507960000000562, 1.2594940000000954, -0.20633800000001284,
   -2.8791000000001006)
   ```
2. Training on the same dataset used to test is a bad idea, as the perceptron is going to learn how to mimic the training data. We will overfit. We may get 100% in training, but when we are exposed to new data, we end up with 50% accuracy, where we’re no better than guessing. We need a separate test and training set to make sure the method learns general trends rather than specifically the training data itself.
   I altered my program to allow for a separate training and test set, and split the dataset up; with 260 training points, and 91 test points. With this dataset, my program converged at around 80% accuracy before overfitting and getting worse results.
   The output at the peak of accuracy was as follows:
   ```
   It took 10000 cycles through the dataset,
   with 2600000 total training calls on the Perceptron
   to reach an accuracy of 80.22%.

   This leaves 17 of the 91 total test instances estimated incorrectly

   The weights the Perceptron used to reach this accuracy were
   (-85.60000000000068, 73.6000000000005, 0.0, 1.540644000002193, -0.47864000000005574,
   29.621448000018088, 5.434232000000433, -16.53762599999283, 22.267597999994383,
   24.733813999987905, 7.21290399999309, -24.53421599998514, -1.748672000000193,
   -3.3247820000031187, 4.439424000000441, 12.358918000004737, -19.218858000001433,
   11.174957999994616, -4.551686000002393, -27.460613999977916, 9.835320000001449,
   21.02787000001031, -1.4552900000003297, -3.1495520000031783, -19.69628000000401,
   4.427213999998897, 3.685862000001237, -27.61895799999853, 6.136485999999641, 30.499909999999133,
   27.920288000004106, 6.33220999999664, 19.088163999992563, -3.477149999999414, -34.8305239999933)
   ```