Assignment 1

Rory Stephenson
300160212

Part 1 - Nearest Neighbour Method

1. K=1 Results:

Classified Iris-setosa as Iris-setosa	Classified Iris-versicolor as Iris-versicolor
Classified Iris-setosa as Iris-setosa	Classified Iris-versicolor as Iris-versicolor
Classified Iris-setosa as Iris-setosa	Classified Iris-versicolor as Iris-versicolor
Classified Iris-setosa as Iris-setosa	Classified Iris-versicolor as Iris-versicolor
Classified Iris-setosa as Iris-setosa	Classified Iris-versicolor as Iris-versicolor
Classified Iris-setosa as Iris-setosa	Classified Iris-versicolor as Iris-versicolor
Classified Iris-setosa as Iris-setosa	Classified Iris-versicolor as Iris-versicolor
Classified Iris-setosa as Iris-setosa	Classified Iris-versicolor as Iris-versicolor
Classified Iris-setosa as Iris-setosa	Classified Iris-versicolor as Iris-versicolor
Classified Iris-setosa as Iris-setosa	Classified Iris-versicolor as Iris-versicolor
Classified Iris-setosa as Iris-setosa	Classified Iris-versicolor as Iris-versicolor
Classified Iris-setosa as Iris-setosa	Classified Iris-versicolor as Iris-versicolor
Classified Iris-setosa as Iris-setosa	Classified Iris-virginica as Iris-virginica
Classified Iris-setosa as Iris-setosa	Classified Iris-virginica as Iris-virginica
Classified Iris-setosa as Iris-setosa	Classified Iris-virginica as Iris-versicolor
Classified Iris-setosa as Iris-setosa	Classified Iris-virginica as Iris-virginica
Classified Iris-setosa as Iris-setosa	Classified Iris-virginica as Iris-virginica
Classified Iris-setosa as Iris-setosa	Classified Iris-virginica as Iris-virginica
Classified Iris-setosa as Iris-setosa	Classified Iris-virginica as Iris-virginica
Classified Iris-setosa as Iris-setosa	Classified Iris-virginica as Iris-virginica
Classified Iris-setosa as Iris-setosa	Classified Iris-virginica as Iris-versicolor
Classified Iris-setosa as Iris-setosa	Classified Iris-virginica as Iris-versicolor
Classified Iris-setosa as Iris-setosa	Classified Iris-virginica as Iris-virginica
Classified Iris-setosa as Iris-setosa	Classified Iris-virginica as Iris-virginica
Classified Iris-setosa as Iris-setosa	Classified Iris-virginica as Iris-virginica
Classified Iris-versicolor as Iris-versicolor	Classified Iris-virginica as Iris-versicolor
Classified Iris-versicolor as Iris-versicolor	Classified Iris-virginica as Iris-virginica
Classified Iris-versicolor as Iris-virginica	Classified Iris-virginica as Iris-virginica
Classified Iris-versicolor as Iris-versicolor	Classified Iris-virginica as Iris-virginica
Classified Iris-versicolor as Iris-versicolor	Classified Iris-virginica as Iris-virginica
Classified Iris-versicolor as Iris-versicolor	Classified Iris-virginica as Iris-virginica
Classified Iris-versicolor as Iris-versicolor	Classified Iris-virginica as Iris-virginica
Classified Iris-versicolor as Iris-versicolor	Classified Iris-virginica as Iris-virginica
Classified Iris-versicolor as Iris-versicolor	Classified Iris-virginica as Iris-virginica
Classified Iris-versicolor as Iris-versicolor	Classified Iris-virginica as Iris-virginica
Classified Iris-versicolor as Iris-versicolor	Classified Iris-virginica as Iris-virginica
Classified Iris-versicolor as Iris-versicolor	Classified Iris-virginica as Iris-versicolor
Classified Iris-versicolor as Iris-versicolor	**69/75 = 92.00% correct**

2. K=3 accuracy = 72/75 (96%)
This improvement in accuracy (4%) is a result of the fact that when you increase the value of k (the number of neighbours to compare to) you decrease the effect of noisy instances in the training set. When using k=1 you simply take the nearest seen example whereas k=3 checks the nearest three which means if the closest instance is an outlier the other two instances can

override it.

3. Pros
   > Simple
   > Easy to use
   > Can achieve good results in many cases

   Cons
   > Have to evaluate every training instance for every classification (inefficient)
   > A classifier isn't learnt per se, more a comparison made

4. In order to apply k-fold (with k=5) classification I would:
   a. Modify my program to accept one data file instead of two
   b. Divide the instances in the data file into five groups
   c. For each group of instances I would treat the group as the test set and the other four groups as the training set and perform the nearest neighbour method
   d. Average the result for each group.

5. I would use K-Means Clustering with k=3 by:
   a. Choose three random instances as means.
   b. Cluster every other instance according to their Euclidean distance from the means (place each instance in the cluster of the mean they are closest to)
   c. Select the centroid of each cluster as the new mean
   d. Re-cluster the instances with the new means
   e. Repeat steps c to d until convergence is reached (either new means are not being selected or a limit number of iterations is exceeded)
6. Run part one runnable jar as: `$ java -jar part1runnable.jar iris-training.data --cluster`

Part 2 - Decision Tree Learning Method

1.

```
Reading data from file hepatitis-training.dat
2 categories
16 attributes
Read 110 instances, base case: live (88/110 80.00%)

Reading data from file hepatitis-test.dat
2 categories
16 attributes
Read 27 instances, base case: live (23/27 85.19%)

Test Results:
Overall: 21/27 = 77.78%
Live: 19/23 = 82.61%
Die: 2/4 = 50.00%

ASCITES = True:
 SPIDERS = True:
  VARICES = True:
   FIRMLIVER = True:
    Class live, prob=1.00
   FIRMLIVER = False:
    BIGLIVER = True:
     STEROID = True:
      Class live, prob=1.00
     STEROID = False:
      FEMALE = True:
       Class live, prob=1.00
      FEMALE = False:
       ANTIVIRALS = True:
        FATIGUE = True:
         Class die, prob=1.00
        FATIGUE = False:
         Class live, prob=1.00
       ANTIVIRALS = False:
        Class die, prob=1.00
    BIGLIVER = False:
     Class live, prob=1.00
  VARICES = False:
   Class die, prob=1.00
 SPIDERS = False:
  FIRMLIVER = True:
   AGE = True:
    Class live, prob=1.00
   AGE = False:
    SGOT = True:
     Class live, prob=1.00
    SGOT = False:
     ANTIVIRALS = True:
      Class die, prob=1.00
     ANTIVIRALS = False:
      STEROID = True:
       Class live, prob=1.00
      STEROID = False:
       Class die, prob=1.00
  FIRMLIVER = False:
   SGOT = True:
```

```
    BIGLIVER = True:
     SPLEENPALPABLE = True:
      Class live, prob=1.00
     SPLEENPALPABLE = False:
      ANOREXIA = True:
       Class die, prob=1.00
      ANOREXIA = False:
       Class live, prob=1.00
    BIGLIVER = False:
     Class die, prob=1.00
   SGOT = False:
    Class live, prob=1.00
ASCITES = False:
 BIGLIVER = True:
  STEROID = True:
   Class die, prob=1.00
  STEROID = False:
   ANOREXIA = True:
    Class die, prob=1.00
   ANOREXIA = False:
    Class live, prob=1.00
 BIGLIVER = False:
  Class live, prob=1.00
```

The decision tree classifier was correct on 77.78% of instances whereas the baseline predictor (live) was correct on 85.19%. The fact that the baseline predictor performed so much better is mainly due to the particular set of instances that were used for testing. If a set of instances in which most were classified as ‘die’ were presented, the baseline predictor would perform far worse whereas the decision tree classifier should perform fairly consistently (assuming it isn’t overfitted, which could be avoided using pruning and a validation set).

This demonstrates the benefit of classification techniques which are general. This means they can perform equally well on unseen sets of instances whereas using the baseline predictor would lead to large fluctuations in accuracy.

2. Using provided generated instances:

Set Name	**Accuracy**	**Fraction**
run01	83.78%	31/37
run02	83.78%	31/37
run03	81.08%	30/37
run04	75.68%	28/37
run05	75.68%	28/37

run06	67.57%	25/37
run07	83.78%	31/37
run08	67.57%	25/37
run09	75.68%	28/37
run10	78.38%	29/37
Average	**77.30%**	**286/370**

3. Pruning:

a. Two ways to prune the tree are:

i. Reduced error pruning: Begging at the leaves you replace each node with its most popular class if doing so does not reduce prediction accuracy (on a validation set).

ii. Cost complexity pruning: Create a set of trees where each tree in the set is created by removing a subtree from the one before it in the set. The subtree to remove at each step is the subtree which causes the least decrease in accuracy of the resulting tree when applied to a validation set. Once the set is generated a cross-validation method or generalised accuracy measure can be used to determine which tree is the best.

b. Pruning decreases the accuracy of the decision tree on the training set because the leaves that are removed represent instances in the training set. Thus the pruned tree has to guess the class of an instance which the un-pruned tree 'knew'. ./part1

c. Pruning may increase accuracy on the test set because it improves the generality of the tree. This is achieved via the fact that the leaves which are removed in the pruning are noisy instances (assuming the pruning method is a good one). Noisy instances are outliers which are not representative of typical instances so removing them makes the resulting tree more likely to classify unseen instances correctly (rather than being influenced by an outlier).

4. The P(A)P(B) purity measure is not a good purity measure when applied to three or more classes because if all instances were split between two classes (out of three or more classes) the purity measure would evaluate to zero which indicates perfect purity despite the fact that the purity is not perfect (all belong to one class). This is a violation of the (required) property of purity measures that they should be minimal (zero) only when all instances belong to one class.

e.g. when 4 instances are split equally between two out of three possible

classes:

$$(0 \div 4) \times (2 \div 4) \times (2 \div 4) = 0$$

Part 3 - Perceptron

NB: I used int's rather than double for my weights in order to reduce memory imprint and on the basis that if we are subtracting/adding ints (the feature values are 1's or 0's) from weights it doesn't make sense for the weights to be doubles in the first place.

1. My perceptron did find a correct set of weights depending on the number of features and maximum number of iterations used. When using 50 features and a maximum of 100 iterations the perceptron typically reaches ~95% accuracy (it varies between execution due to the fact that random features are selected). When using more features or a higher number of iterations the perceptron would consistently reach 100% accuracy on the training set (200 features and 100 maximum iterations or 50 features and 500 maximum iterations).

   Even when small numbers of features and iterations were used (<10 for both) an accuracy over 90% could be achieved. This is a result of the fact that the weights/features are being tested on the same set they were generated with so they are probably extremely overfitted. Essentially with a small number of features/weights the perceptron is just dividing the set into true/false results rather than creating a classifier which can predict the class of an unseen instance based on a combination of its features.

2. Evaluating the perceptron its training data is a bad idea because there is no way to know when the perceptron is becoming overfitted since the more overfitted it becomes, the better it will be at classifying the training set. The result is that the resulting perceptron will not be general (the results for unseen instances vs seen instances will be much worse).

   I did not generate extra image data because the code for generating them was broken however I would expect the accuracy to be far lower since the perceptron will be strongly fitted to the training data. To alleviate this problem a validation set could be used and once the perceptron begins to produce lower accuracies on the validation set, training could be stopped.