COMP307 – Assignment 1
greenthom – 300536064

kNN

Have normalized by KNN algorithm to improve accuracy.

3.a. Report the classification accuracy on the test set using k=1 and k=3. Include the output files corresponding to each experiment.

```
tomgreen@Toms-MacBook-Pro src % java data_part1.kNN data_part1/wine_train.csv data_part1/wine_test.csv data_part1/knn1_test.txt 1
Correct Prediction: 28.0 out of 36
kNN Accuracy: 77.77777777777779%
```

K=1 is saved under knn1_test.txt with accuracy of 77. 77777777777779%
As you can see that in the case of the knn when K = 1, there are 28 correct classified instances out of 36 test instances.

```
tomgreen@Toms-MacBook-Pro src % java data_part1.kNN data_part1/wine_train.csv data_part1/wine_test.csv data_part1/knn3_test.txt 3
Correct Prediction: 29.0 out of 36
kNN Accuracy: 80.55555555555556%
```

K=3 is saved under knn3_test.txt with accuracy of 80.55555555555556%
As you can see that in the case of the knn when K = 3, there are 29 correct classified instances out of 36 test instances.

3.b. Report the classification accuracy on the train set using k=1 and k=3. Include the output files corresponding to each experiment.

```
tomgreen@Toms-MacBook-Pro src % java data_part1.kNN data_part1/wine_train.csv data_part1/wine_train.csv data_part1/knn1_train.txt 1
Correct Prediction: 142.0 out of 142
kNN Accuracy: 100.0%
```

K=1 is saved under knn1_train.txt with an accuracy of 100%
As you can see in the case of knn when K=1, there are 142 correct classified instances out of 142 test instances.

```
tomgreen@Toms-MacBook-Pro src % java data_part1.kNN data_part1/wine_train.csv data_part1/wine_train.csv data_part1/knn3_train.txt 3
Correct Prediction: 125.0 out of 142
kNN Accuracy: 88.02816901408451%
```

K=3 is saved under knn3_train.txt with accuracy of 88.02816901408451%
As you can see in the case of knn when K=3, there are 125 correct classified instances out of 142 test instances.

The reason for this is in the case of the training set, the data is placed on top of itself essentially. If we use a K value of 1 to find the nearest neighbor, it will be itself since all the test values and their nearest neighbors are on top of each other.

3.c. When analyzing the results from the test and train sets using k=1 and k=3, consider the following questions:

i. Does a specific k value maximize accuracy in the training set? If yes, does this optimal k value also improve accuracy in the test set?

The k value of k=1 does maximize the accuracy in the training set to 100% accuracy. However, this k=1 value does not improve the accuracy in the test set with it having an accuracy of 77. 77777777777779%

ii. Why does using this optimal k value (in the train data) work well or not work well for the test set?

Using a k value of 1 in KNN on the training data (using training data for both training and test), every prediction will be correct. This is because each instance in the test set finds its identical neighbor in the training set which results in a perfect accuracy.

The reason that the k value of 1 works well in test data is because it contains instances that are very similar to instances in the training set. Therefore, using a k value of 1 can still have high accuracy on the test set due to the proximity of instances between train and test datasets. This may not result well when tested with unseen data (seen by using wine_train and wine_test) due to overfitting which occurs when the model learns the training data well. This includes random fluctuations and noise which may not exist in the test data.

3.d. Discuss the consequences of increasing and decreasing k on the algorithm's performance. Additionally, consider scenarios where k is large, for example, k equals the total size of the dataset. How do these extreme values of k influence the behavior and effectiveness of the kNN algorithm?

With a high value for k, matching the size of the training data, the algorithm considers all points in the dataset as potential neighbors for each prediction. This potentially leads to a better generalization of unseen data. Consequently, predictions are made based on a majority vote among all training instances. This may lead to smoothing and underfitting which can result in the model failing to interpret patterns in the data. The class label that appears most frequently among the k nearest neighbors is assigned to the query point. By setting k to 142 (size of the training data), we find that the accuracy when testing the wine_test.csv data is 38.88888888888889%.

```
tomgreen@Toms-MacBook-Pro src % java data_part1.kNN data_part1/wine_train.csv data_part1/wine_test.csv data_part1/output.txt 142
Correct Prediction: 14.0 out of 36
kNN Accuracy: 38.88888888888889%
```

This indicates that 38.88888888888889% of the classes in the test data match the most common class in the wine_train.csv data. Overfitting is far more likely to occur as the model is more likely to capture neighbors that distract from the correct result. If we have a high k value every point in the dataset will contribute to the decision process meaning the boundaries will be much smoother. Reducing the value of k could result in a more complex decision boundary, which allows it to capture finer distinctions within the data. However, this can also heighten the susceptibility to noise and outliers which can result in overfitting.

Decision Tree

3.a. Report the classification accuracy for "rtg_A" and the decision tree output. Include the output file corresponding to the decision tree.

```
tomgreen@Toms-MacBook-Pro src % java data_part2.DecisionTree data_part2/rtg_A.csv data_part2/DT_A.txt
DT Output to: data_part2/DT_A.txt
DT Accuracy is: 100.0%
```

Have 100.0% accuracy for rtg_A.

3.b. Report the classification accuracy for "rtg_B" and the decision tree output. Include the output file corresponding to the decision tree.

```
tomgreen@Toms-MacBook-Pro src % java data_part2.DecisionTree data_part2/rtg_B.csv data_part2/DT_B.txt
DT Output to: data_part2/DT_B.txt
DT Accuracy is: 100.0%
```

Have 100.0% accuracy for rtg_B.

3.c. When analyzing the results for rtg_A and rtg_B, consider the following questions:

i. Which features result in the minimal entropy when considering the entire training dataset? How can this observation be made?

rtg_A:

```
feature 0 (IG: 0.11667318106592722, Entropy: 0.40217919020227283)
```

rtg_B:

```
-- feature 0 == 0 --
    feature 2 (IG: 0.12296085056279982, Entropy: 0.48977901368693755)
    -- feature 2 == 0 --
```

This observation can be made by splitting on features with the highest information gain. From here we can identify the feature with the lowest entropy. The one with the lowest entropy will have the least uncertainty or randomness in predicting the class labels. This feature provides the most IG and is considered the best feature for splitting the dataset.

rtg_A: Feature 0 results in the lowest entropy. This is found by looking at the first feature split. The algorithm chose feature 0 first meaning it will give the highest IG therefore it will result in the lowest entropy at that point.

rtg_B: Feature 2 would result in lowest entropy if it was the only to use which can be seen by looking at the split. Since it chose feature 2 it means it provides the highest IG which would result in the lowest entropy at that point.

ii. Were there any features that were not utilized in constructing the tree for both datasets? If so, what factors contributed to their exclusion from the tree-building process?

In rtg_A: Feature 3 was also not used in rtg_A due to its features being 0 for every point which meant that it does not contain any information. If we tried to split this feature, we would not change the dataset at all there.

In rtg_B: all features were used in the tree-building process due to it having varying values of 0 and 1.