2802ICT Intelligent Systems
Assignment 2 Task 1 Report

By Nathan Cowan
S5143344

Software Design:
The load() function takes no arguments, it's only purpose is to read the 'iris.csv' file into a workable format and return it. The first 4 values are taken directly as floats, but the species is converted into an int where 1 = 'setosa', 2 = 'versicolor' and 3 = 'virginica'. This is done because ints are much easier to compare and interact with and the species names themselves do not actually contain any information so nothing is lost by changing them to another arbitrary classification, as long as they remain separate.

Split_data() works exactly as described in the task sheet. It takes the dataset and at random splits the data into training and testing. Both the training and testing datasets are returned.

sortKey() is used for sorting data by distance in the knn() function. It takes an element in the form of a tuple of a flower's value and its distance to the flower being tested and returns the distance.

kNN() takes a training set, a sample flower and a k value as arguments. It calculates the distance from the sample flower to each flower in the training set and returns the k closest as determined by the distance function.

The euclidean_distance() function is as it's named. Given two flowers it will calculate the euclidean 'true' distance over all variables except for 'species' and return the result.

For the function alternative_distance() I used the plots shown in figures 1 to 4 to visualise the relative ranges of each species in order to see which variables overlap the least between species. According to those plots the petal_length and petal_width are the most distinguishing features as sepal_length and sepal_width overlap much more heavily between different species. alternative_distance() because of this calculates distance by weighting the variables sepal_length and sepal_width with a *0 multiplier. The difference between two values is also summed rather than squared. (*Note that although this is implemented as only comparing petal_length and petal_width this is equivalent to weighting the other variables by *0*)

The classifier() function much like the kNN() function takes a training dataset, a sample to test, and the k value, and runs kNN() on those same inputs. For each of the nearest neighbors returned by kNN() their species is summed and then divided by k to ascertain the mean. As this is not integer division the resulting float is then rounded to the nearest integer and then returned as the predicted species.

Lastly the accuracy() function which takes the input file path, the % ratio and the k value. Other information such as the dataset itself is loaded automatically inside the function using load(). After the data is loaded it is split and each flower is classified before the total accuracy is calculated and returned.

Results:

Figure 5 shows the accuracy results for values of k from 1 to 150 for a naive euclidean distance function which simply sums the difference in each variable between two flowers. It can be seen that the most accurate k values are less than ~55 after which it plateaus at ~⅓ as it approaches 80. The plateau can be explained easily by the fact that there are only 50 to a species, this means that even if all of 50 nearest neighbors are of the correct species any more than that increases the proportion of neighbors that are a different species until it is essentially choosing from guessing between 2 species. The same thing happens further as it reaches all three species as k approaches 150.

Before this happens at k values less than ~55, you can still see that the accuracy is steadily dropping, which simply indicates that by this distance function the further away the neighbors being considered the more likely it is that they are a different species, which suggests that for this dataset the sum of the differences in values is actually an effective means of evaluation.

As such the most effective k value is 1, which although devolving the algorithm into simply the nearest neighbor, happens to be the best for this distance function and dataset at ~95% accuracy.

Figure 6 shows the accuracy results for values of k from 1 to 150 for an alternative distance function which instead sums the difference in only the petal_length and petal_width between two flowers as these are the most relevant variables for determining species.

This shows exactly the same plateau at ~⅓ accuracy at the same values as in figure 5. The difference is in the range of 1-55 where both distance functions are still somewhat effective. Figure 6 shows that although the accuracy is still rapidly decreasing by k=55 it is notably more stable as k increases from 1 than in figure 5 now that the sepal_length and sepal_width values are not causing any distraction.

Figure 7 simply shows an example of this program's output, specifically it shows several classifier predictions and results as well as the accuracy for that group both as a fraction and a percent.

Conclusions:

In both cases a k value of 1 is the most accurate at ~95% so it seems that for this dataset nearly any sane distance function will have similar properties and that the nearest neighbor is the most significant by far in predicting the species of any given flower.

Appendix:

Plots and graphs:

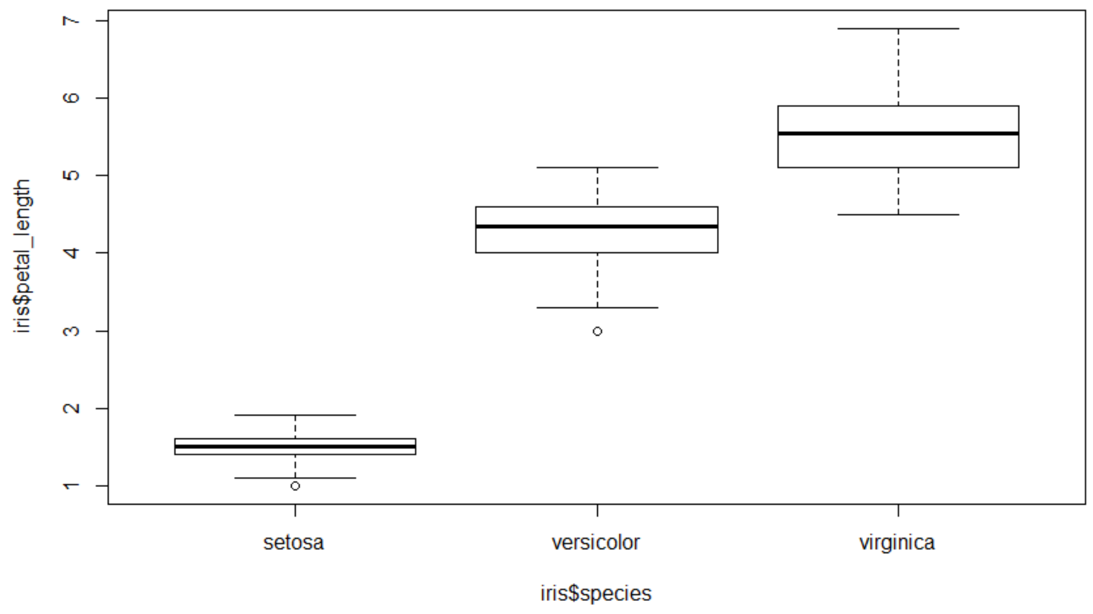


Figure 1: box plot of petal_length separated by species.

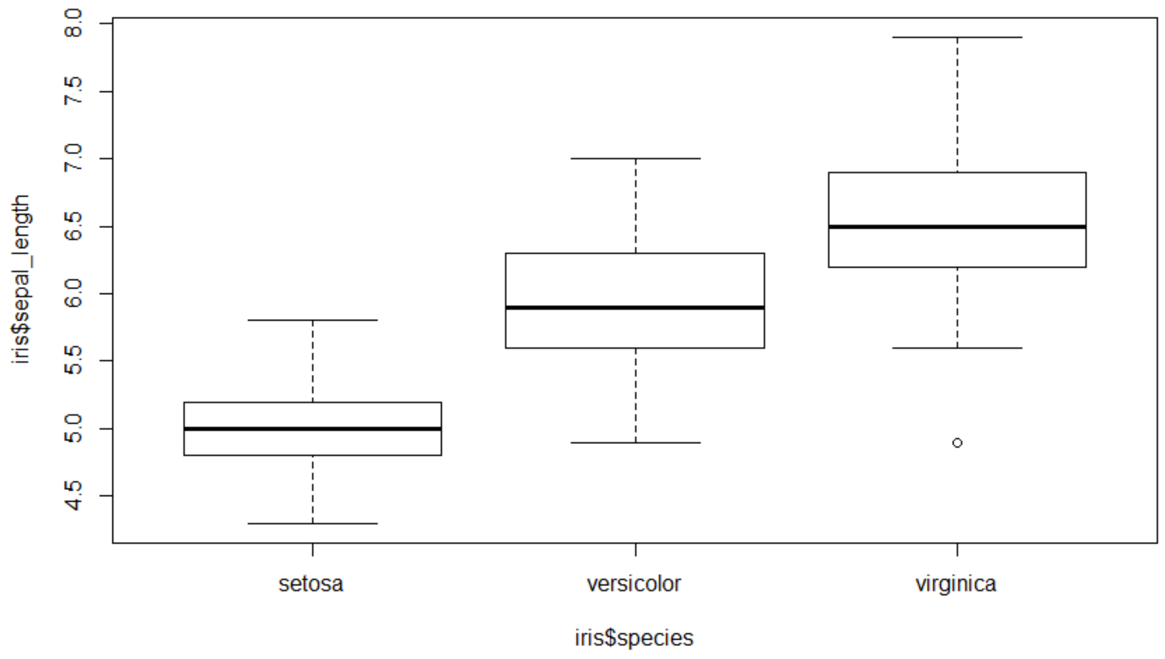


Figure 2: box plot of sepal_length separated by species.

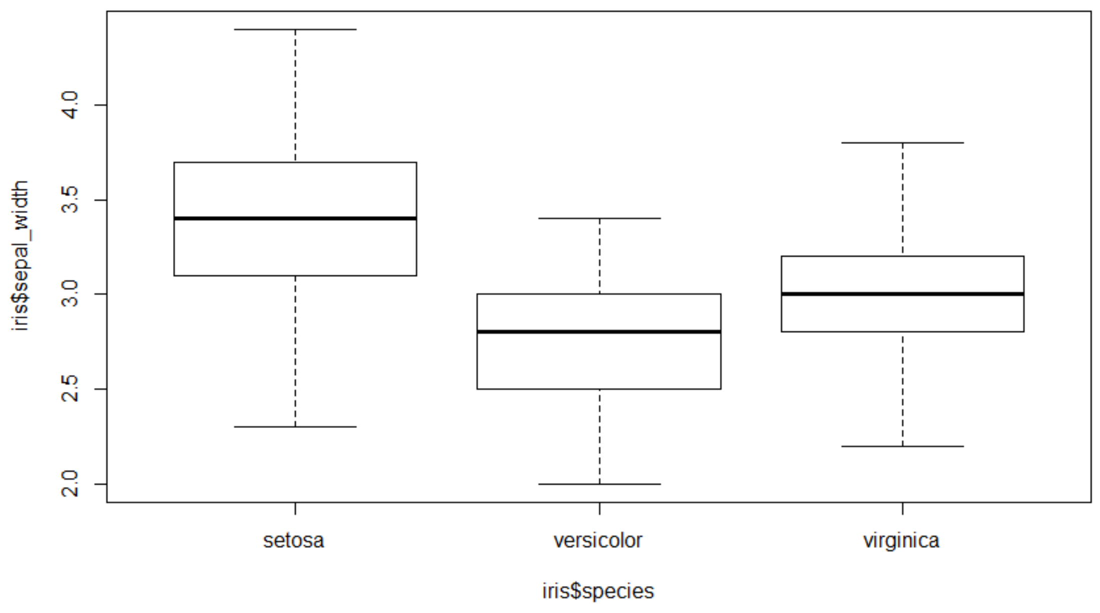


Figure 3: box plot of sepal_width separated by species.

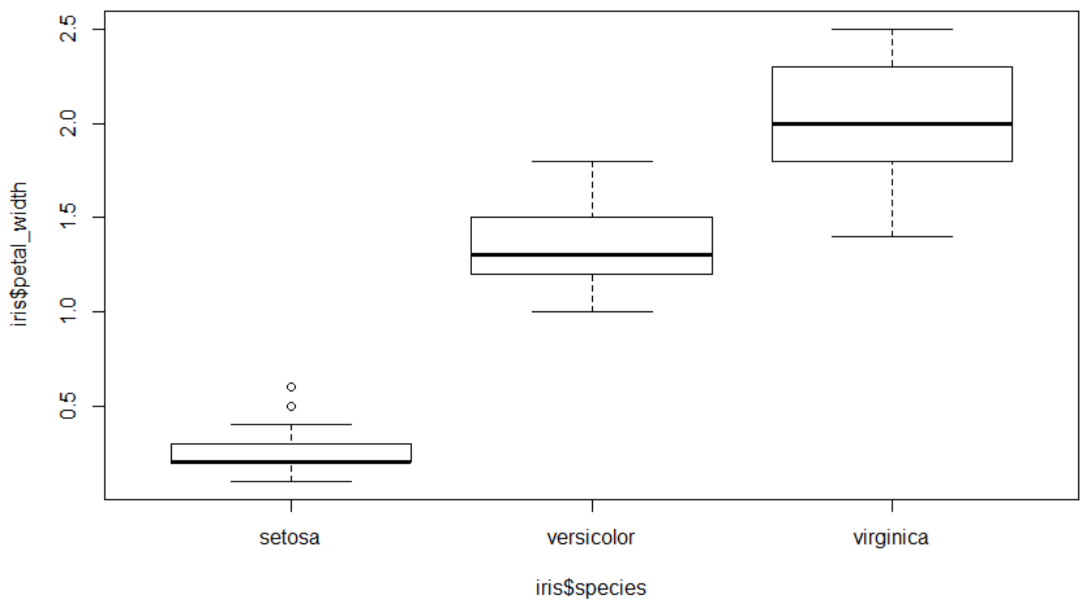


Figure 4: box plot of petal_width separated by species.

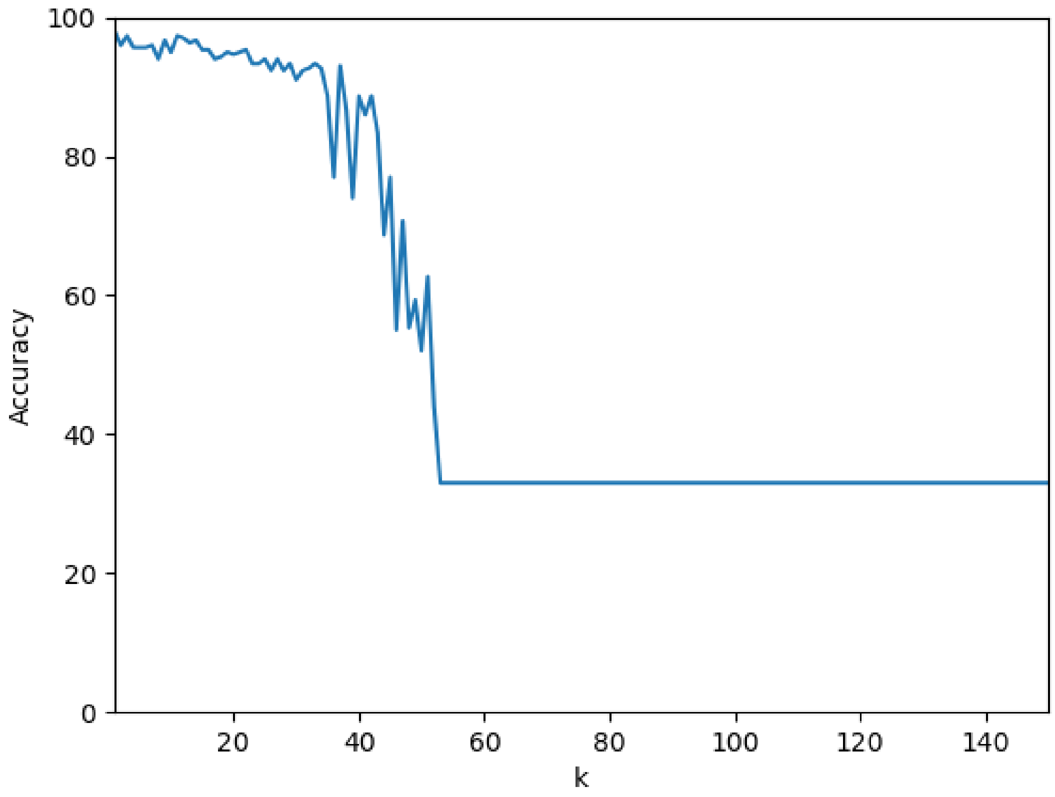


Figure 5: accuracy of different K-values using euclidian_distance function.

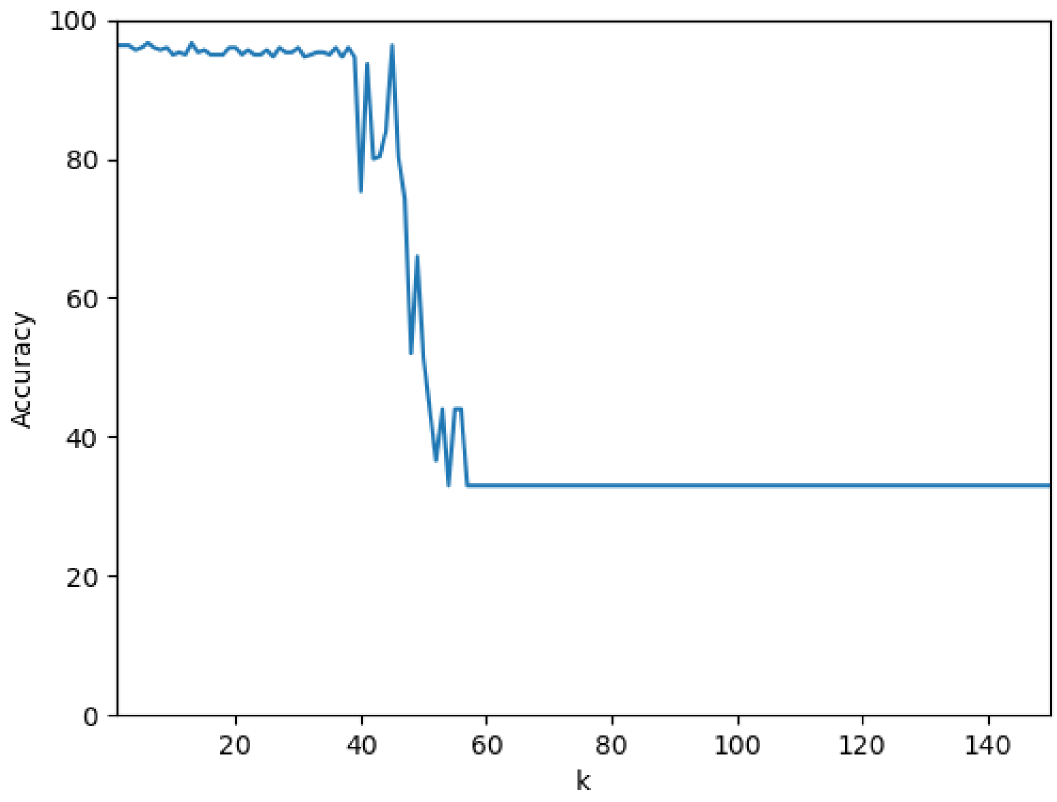
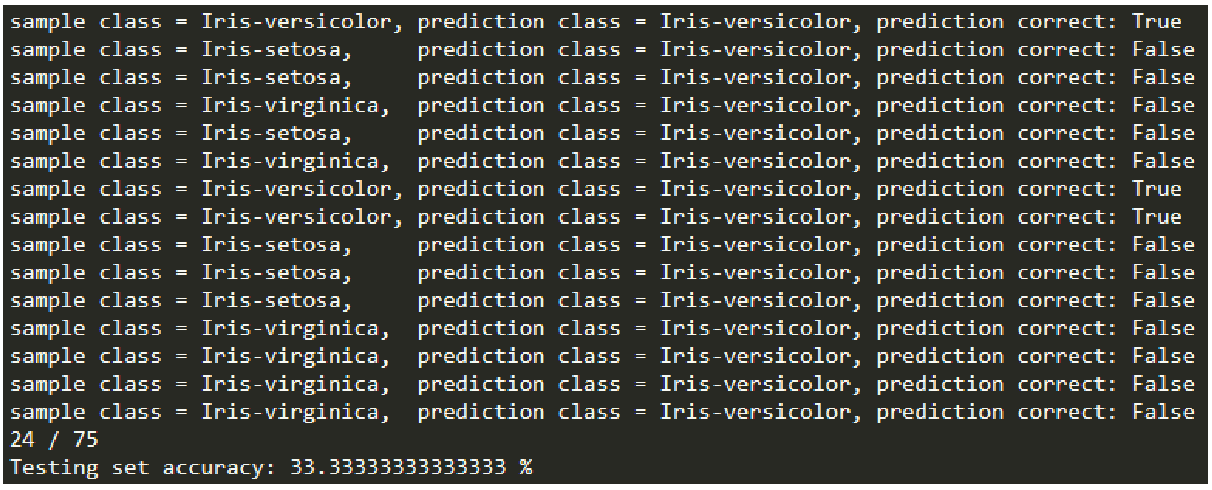


Figure 6: accuracy of different K-values using alternative_distance function



```
sample class = Iris-versicolor, prediction class = Iris-versicolor, prediction correct: True
sample class = Iris-setosa,     prediction class = Iris-versicolor, prediction correct: False
sample class = Iris-setosa,     prediction class = Iris-versicolor, prediction correct: False
sample class = Iris-virginica,  prediction class = Iris-versicolor, prediction correct: False
sample class = Iris-setosa,     prediction class = Iris-versicolor, prediction correct: False
sample class = Iris-virginica,  prediction class = Iris-versicolor, prediction correct: False
sample class = Iris-versicolor, prediction class = Iris-versicolor, prediction correct: True
sample class = Iris-versicolor, prediction class = Iris-versicolor, prediction correct: True
sample class = Iris-setosa,     prediction class = Iris-versicolor, prediction correct: False
sample class = Iris-setosa,     prediction class = Iris-versicolor, prediction correct: False
sample class = Iris-setosa,     prediction class = Iris-versicolor, prediction correct: False
sample class = Iris-virginica,  prediction class = Iris-versicolor, prediction correct: False
sample class = Iris-virginica,  prediction class = Iris-versicolor, prediction correct: False
sample class = Iris-virginica,  prediction class = Iris-versicolor, prediction correct: False
sample class = Iris-virginica,  prediction class = Iris-versicolor, prediction correct: False
24 / 75
Testing set accuracy: 33.33333333333333 %
```

Figure 7: program output example.