Homework 1

Kevin Ding

September 12 2022

Problem 1

(a): See q1.py file for this question.

(b): See q1.py file for this question.

(c): See q1.py file for this question.

(d): The exact time of each method varies slightly every time I run it, but the difference between them generally stays at the same level. The run time of the non-vectorized approach (for loop) is roughly 0.76 seconds, whereas the run time of the vectorized approach using the Numpy package is roughly 0.0005 seconds. The vectorized approach is roughly a thousand times (1520 to be more exact in this case) faster than the for loop approach.

Problem 2

(a): See q2.py file for this question.

(b): Figure 1 below is the boxplot of the distribution of sepal length of the species type.

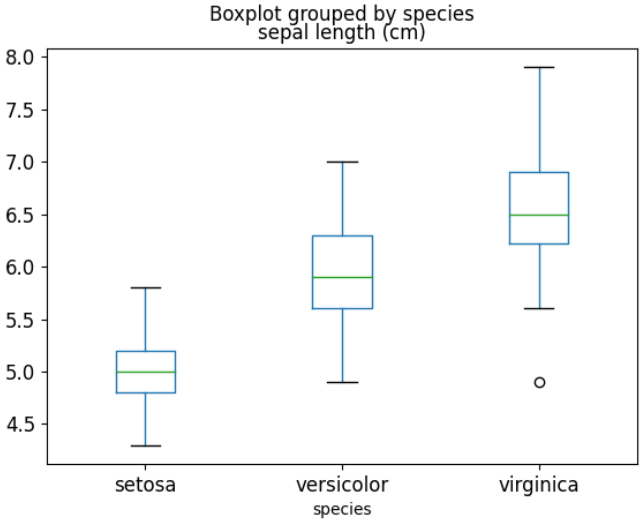

Figure 1

Figure 2 below is the boxplot of the distribution of sepal width of the species type

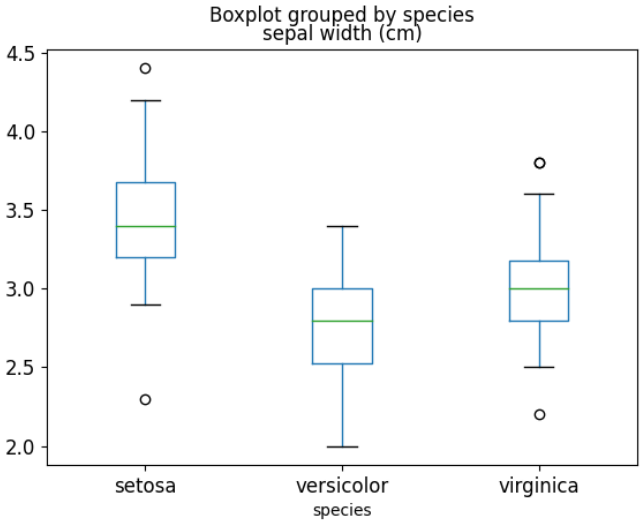

Figure 2

Figure 3 below is the boxplot of the distribution of petal length of the species type

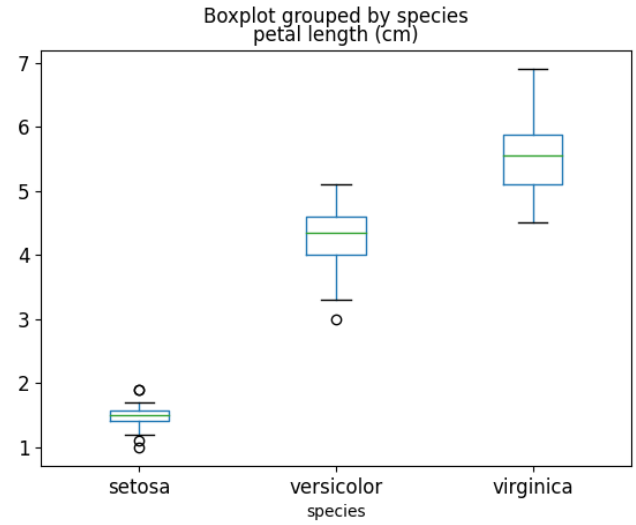

Figure 3

Figure 4 below is the boxplot of the distribution of petal width of the species type

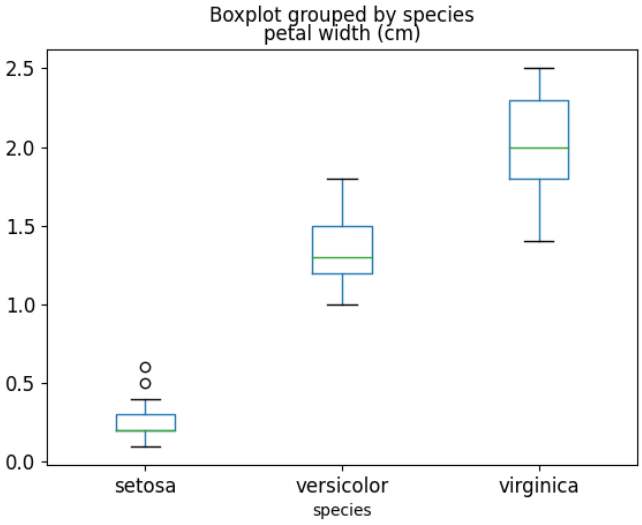

Figure 4

(c): Figure 5 below is the scatterplot of the sepal width and sepal length colored by species.

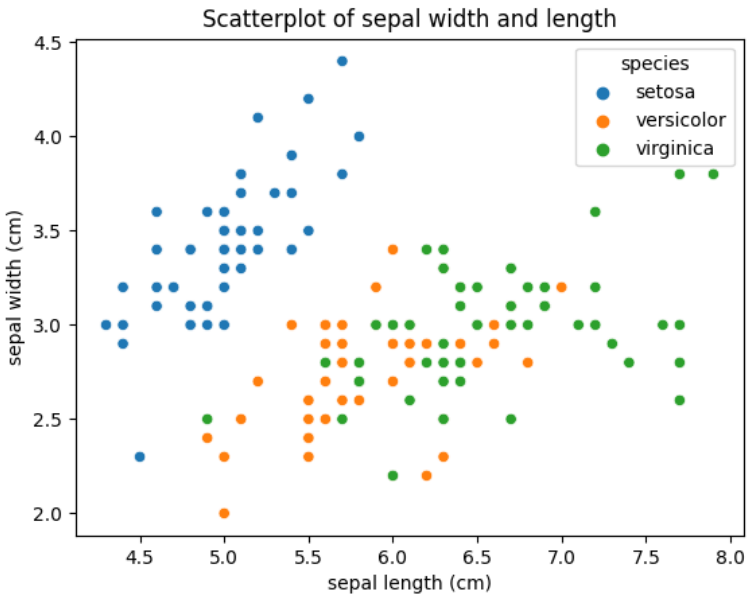

Figure 5

Figure 6 below is the scatterplot of the petal width and petal length colored by species.

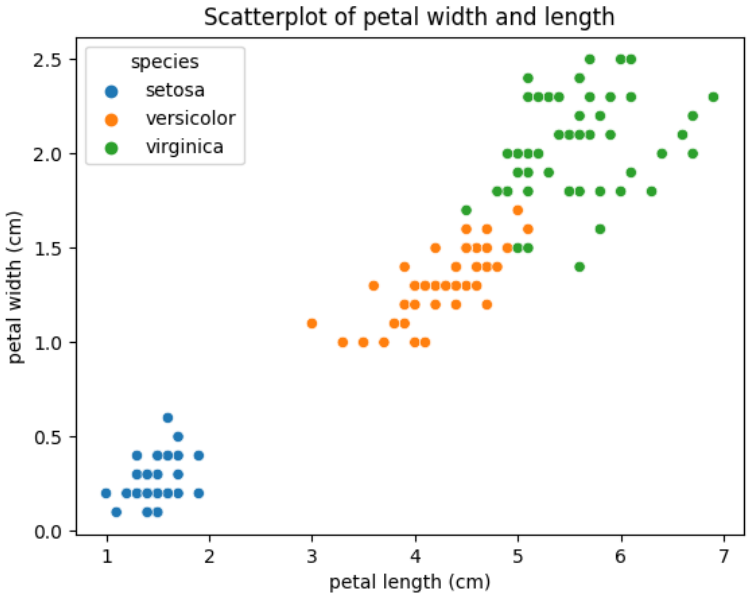

Figure 6

(d): The two scatterplots have different distributions. The sepal scatterplot is bit hard to distinguish the species, as we can see the versicolor and virginica species mix together. However, the petal scatterplot is much easier to recognize individual species.

To predict future points, we can use petal attributes since it is easier to spot species. First look at petal length, if the petal length $< 2.5cm$, then we immediately classify the point to be of setosa. If the petal length $\geq 2.5cm$, then we check if the petal length $< 5cm$. If the petal length is $< 5cm$, we classify the point to be of versicolor. If the petal length is $\geq 5cm$, then we check its petal width. If the petal width $< 1.75cm$, then we classify the point to be of versicolor. If the petal width $\geq 1.75cm$, then we classify the point to be of virginica.

To summarize, consider below

petal length $< 2.5cm$: setosa
petal length $\geq 2.5cm$ but $< 5cm$: versicolor
petal length $\geq 5cm$ but petal width $< 1.75cm$: versicolor
petal length $\geq 5cm$ but petal width $\geq 1.75cm$: virginica

Problem 3

(a): See knn.py file for this question.

(b): See knn.py file for this question.

(c): See knn.py file for this question. By running this file, we can tell that the training accuracy for $k = 1$ is 1 and the training accuracy for $k = 2$ is 0.953, meaning that 95.3% of the samples have been correctly classified. For testing accuracy on the other hand, for $k = 1$, the accuracy is 0.903 and for $k = 2$, it is about 0.892.

(d): For both training accuracy and testing accuracy plots, I have generated k from 1 up to 100. Allowing up to $k = 100$ allows us to see a general trend of how the accuracy changes. For $k100$, it is very likely the general trend is the same or similar.

The red line in figure 7 below is the training accuracy of different k values. From the graph we can see the training accuracy is 1 for $k = 1$. For $1 \leq k \leq 20$ roughly, the training accuracy is approximately somewhere between 0.92 and 0.94. For $k20$, we can see that the trend stays the same, with no big change both up and down. The training accuracy is usually somewhee between 0.92 and 0.93 for this range.

The blue line in figure 7 below is the testing accuracy of different k values.

From the graph we can see that the testing accuracy seems to have slightly more variance than the training accuracy. The testing accuracy varies quite a bit in the first few k values. For instance, when $k = 2$, the testing accuracy is roughly 0.89, and then it bounces back to roughly 0.925 when $k = 5$. When $k \geq 20$, the testing accuracy shifts between 0.915 to 0.93. In general, the training accuracy seems to be slightly greater than the testing accuracy.

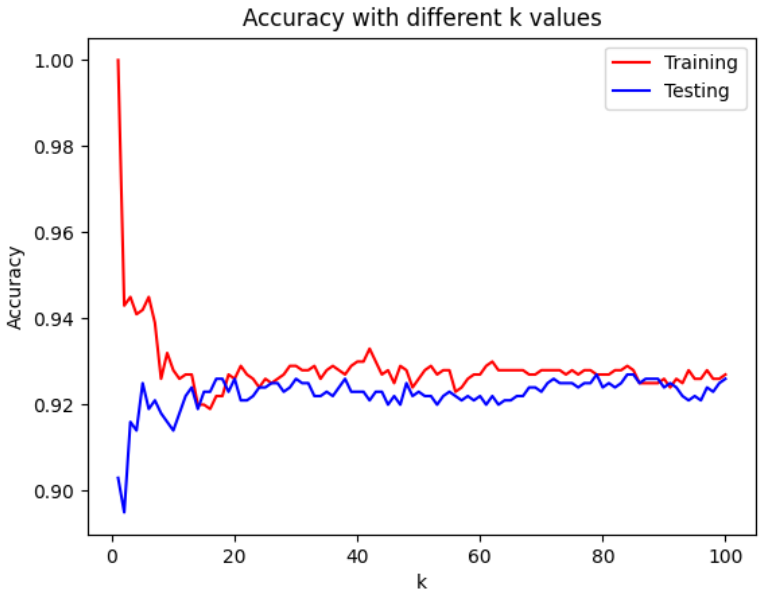

Figure 7

(e): My predict function implementation's time complexity is $O(nd+nlgn+k)$ for predicting one new point, namely one testing point. In my predict function, I first check the distances between the testing point to each training point in the training set, with d features. I linearly loop through the training set, so the time complexity for this part is $O(nd)$. After I get the distance list for this one testing point, I then use quick sort to sort the distance list based on the distance, which takes $O(nlgn)$ on average. In the very unlikely worst case, this step would take $O(n^2)$ if the list is sorted already. After I sort the list, next step is to traverse the first k elements of the list to compute the simple majority value, and this would take $O(k)$. Therefore, overall the time complexity is $O(nd + nlgn + k)$.

Problem 4

(a): See q4.py file for this question.

(b): See q4.py file for this question.

(c): See q4.py file for this question.

(d): Figure 8 below is the plot of the accuracy of different preprocessing techniques as a function of k from 1 to 100. We can see that for $k \leq 10$, the accuracy of 4 techniques varies a lot. For $20 \leq k \leq 80$ roughly, standard scale and min max scale accuracy is greater than the ones with no-preprocessing and irrelevant features. From $k80$, the accuracy of standard scale and min max scale seem to decrease gradually and at $k = 100$, four techniques' accuracy are relatively similar or close to each other, all at somewhere between 0.86 and 0.87. Another thing to notice is that for no preprocessing and irrelevant features, the accuracy does not seem to vary a lot starting from roughly $k \geq 30$, meaning that probably the simple majority value stays the same for $k \geq 30$. For standard scale and min max scale, the gradual decrease in accuracy from $k \geq 60$ roughly might be because of overfitting value, meaning we try to memorize too much of the training set and it does not generalize that well to the new test set.

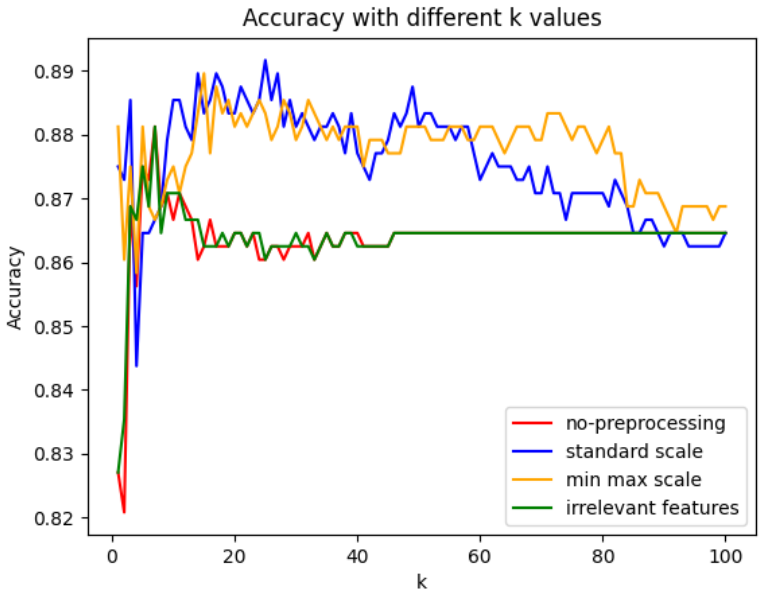

Figure 8