CS 334 - Homework 1

Alex Welsh

September 7th 2020

1 Question 1 - Numerical Programming

The vectorized approach was much faster compared to the for loop method of calculation. It was 0.0015s vs 1.3261s for the vectorized and for loop methods, respectively.

See q1.py for the code to calculate the sum of squares.

2 Question 2 - Visualization Exploration

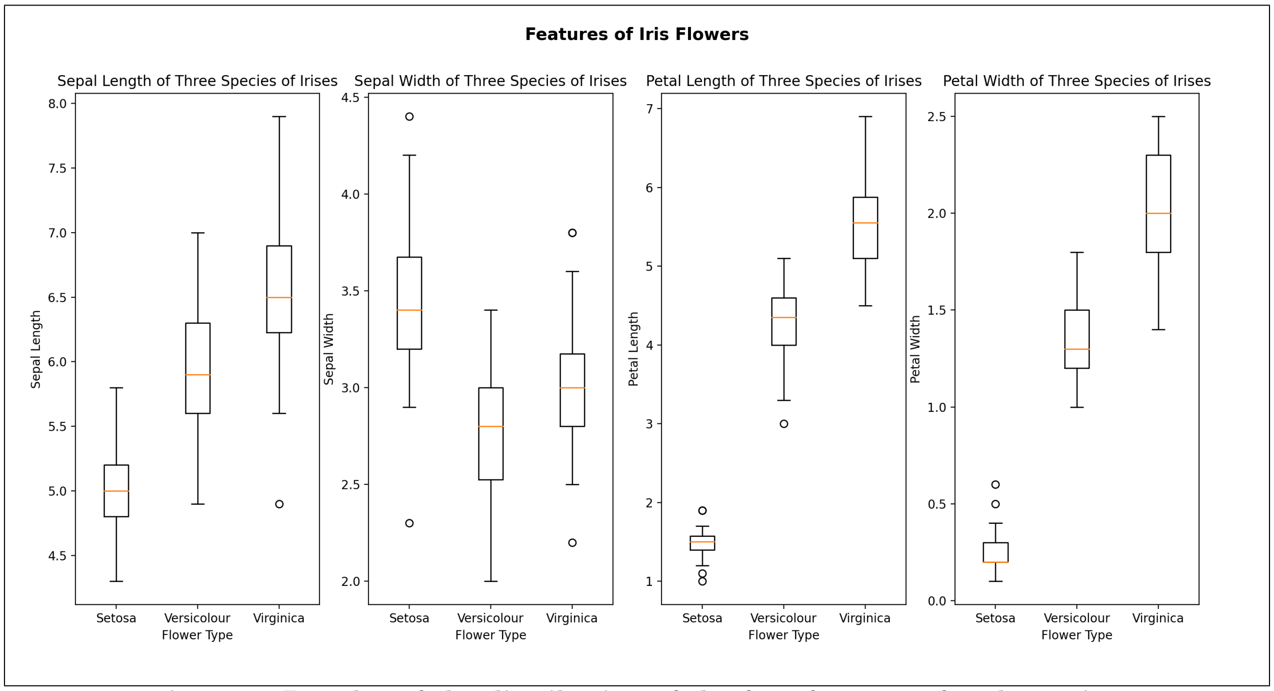

Figure 1. Boxplot of the distribution of the four features of each species.

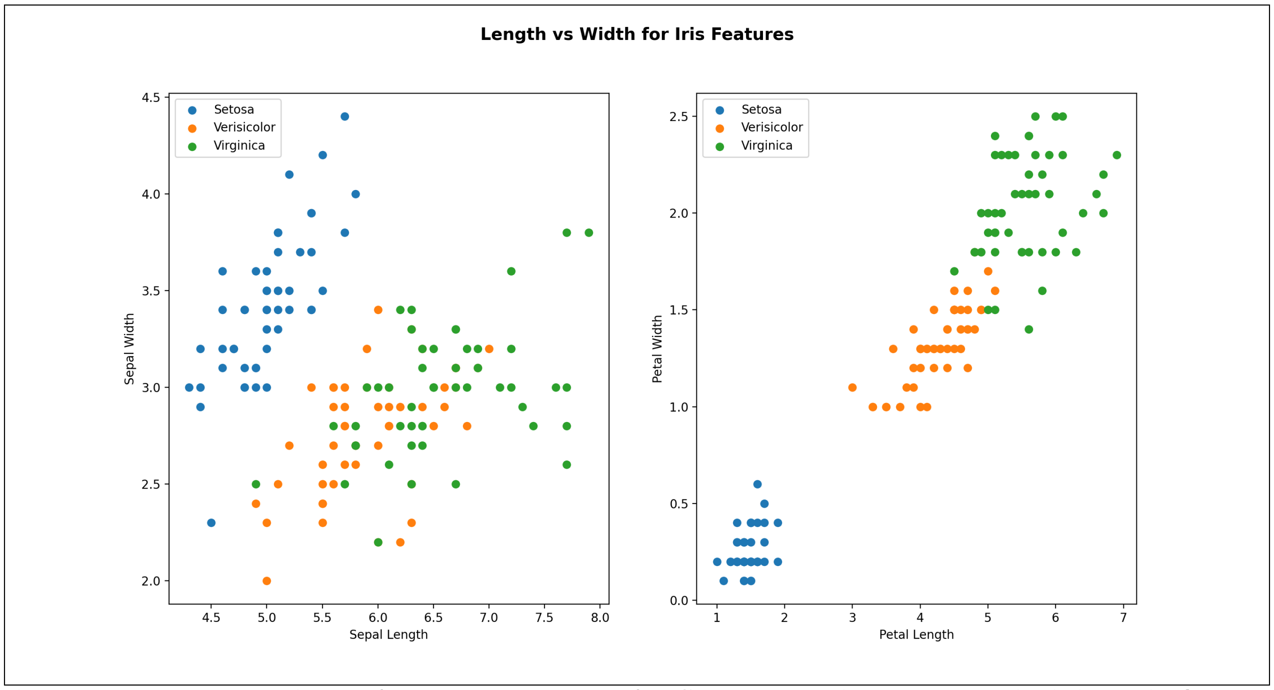

Figure 2. Length vs width of petals and sepals for Setosa, Versicolor, and Virginica Iris flowers.

Based on Figure 1, Setosa's can be identified by their small petal length and width. They also have a smaller sepal length paired with a longer sepal width based on Figure 2. Verisicolor and Virginica are harder to separate. Using the figure from Figure 1, Virginica's tend to have longer sepal length, petal length, and petal width. They also have the longest petal length-petal width pair (Figure 2). The rest are likely to be Verisicolor and have the middle values of petal length and width (Figure 1, 2).

See q2.py for the code that generates the plots for question 2b (Figure 1) and question 2c (Figure 2).

3 Question 3 - K-NN Implementation

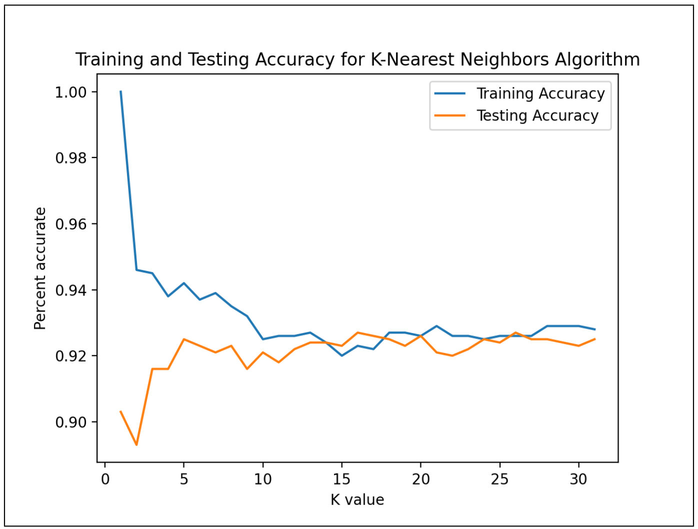

Figure 3. Training and testing accuracy of KNN for K=1 to K=31. The highest test data accuracy was 92.7% for K=16 and 26.

The computational complexity of the KNN is $O(nd + nlog(n) + k)$ because for n training samples, d features are used for each n to compute the distance between the training sample and the current test sample (nd). Then, n distances need to be sorted. My implementation used quicksort which has an $nlog(n)$ runtime. Then, the k closest samples are used to determine the class. Thus, adding them together gives a complexity of $O(nd + nlog(n) + k)$. They are added because each step is performed sequentially.

See knn.py for the code that runs the KNN algorithm for the test data. The performance method runs the KNN algorithm from K=1 to K=args.k to generate the plot shown (Figure 3).

4 Question 4 - K-NN Performance

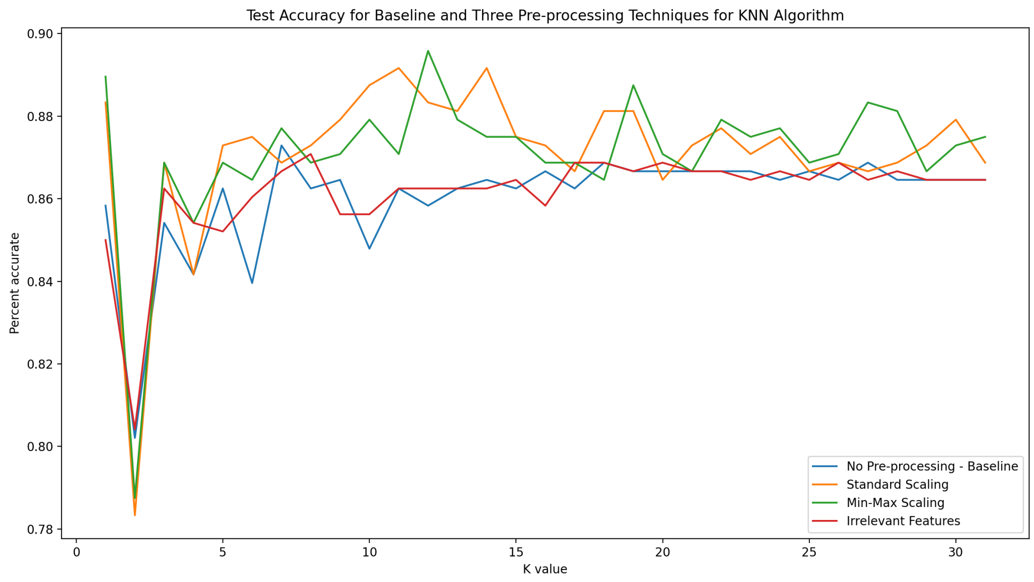

Figure 4. Testing accuracy of KNN for K=1 to K=31 with 3 types of pre-processing.

The standard and min-max scaling appear to improve the performance of the KNN, especially for K between 8 and 15 when compared to the baseline (no pre-processing) and the addition of two irrelevant features. The addition of two irrelevant features performs almost exactly the same as the baseline model. The two types of pre-processing used improve the model accuracy to 89.1% at K=11 for standard scaling and 89.6% at K=12 for min-max scaling. Thus, these seem to be the ideal values of K and types of pre-processing to produce the best model for this data set.

See q4.py for the code that runs the different types of pre-processing and the for loop that compares the baseline (no pre-processing) to the three types of pre-processing from K=1 to K=args.k.