Programming Assignment 1

Jake Sciotto
EN685.621 - Algorithms for Data Science
770-286-8893
Johns Hopkins University

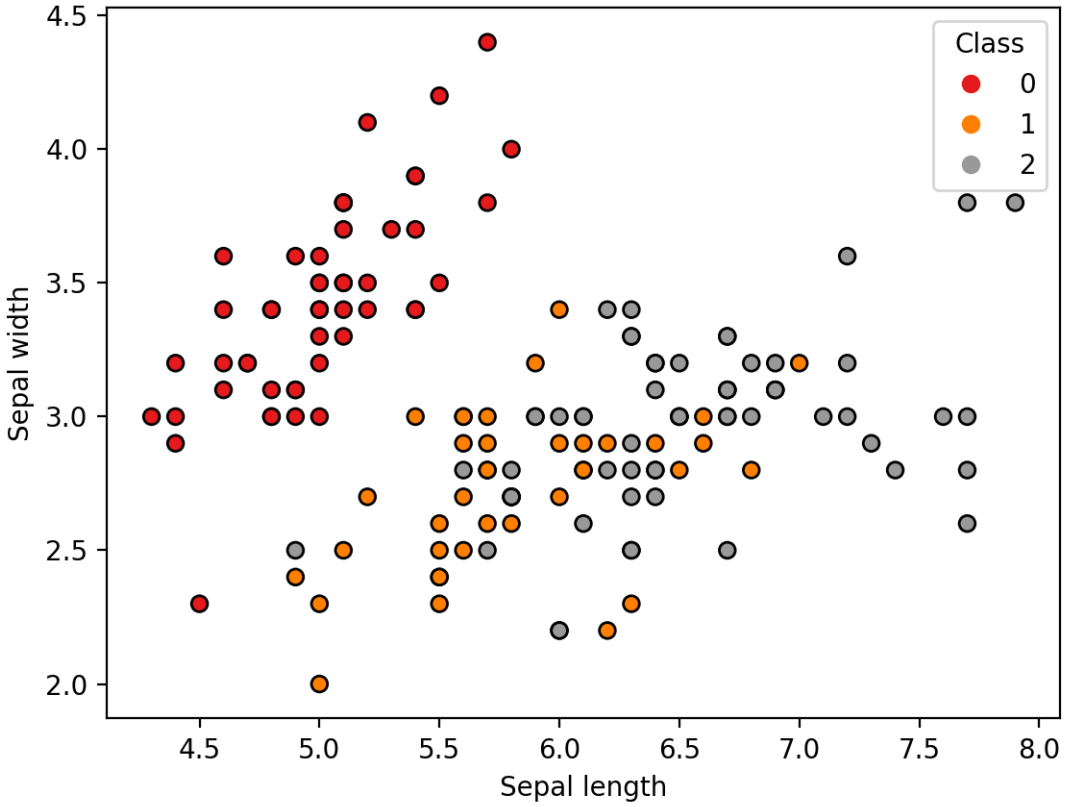
July 28, 2020

1. Preparation

   To prepare the Iris data, I imported the dataset from the `sklearn.datasets` module of Python.

2. Visualization

   Next, we looked at two different features of the newly imported data. Below is sepal-length plotted against sepal-width, separated by class.



3. Sorting

   To perform a sort, I chose a simple insertion sort to attempt to separate classes. Every time that an exchange is made, the row is swapped as well. From this I surmised that ample separation could be achieved by only two features: petal-length and petal-width. There is still some noticeable overlap between versicolor and virginica values between petal-length of 4.5 - 5.1, and petal-width of 1.4 - 1.8.

```
[6.0 3.4 4.5 1.6 1 'versicolor']
[4.9 2.5 4.5 1.7 2 'virginica']
[6.5 2.8 4.6 1.5 1 'versicolor']
[6.6 2.9 4.6 1.3 1 'versicolor']
[6.1 3.0 4.6 1.4 1 'versicolor']
[7.0 3.2 4.7 1.4 1 'versicolor']
[6.3 3.3 4.7 1.6 1 'versicolor']
[6.1 2.9 4.7 1.4 1 'versicolor']
[6.1 2.8 4.7 1.2 1 'versicolor']
[6.7 3.1 4.7 1.5 1 'versicolor']
[5.9 3.2 4.8 1.8 1 'versicolor']
[6.8 2.8 4.8 1.4 1 'versicolor']
[6.2 2.8 4.8 1.8 2 'virginica']
[6.0 3.0 4.8 1.8 2 'virginica']
[6.9 3.1 4.9 1.5 1 'versicolor']
[6.3 2.5 4.9 1.5 1 'versicolor']
[5.6 2.8 4.9 2.0 2 'virginica']
[6.3 2.7 4.9 1.8 2 'virginica']
[6.1 3.0 4.9 1.8 2 'virginica']
[6.7 3.0 5.0 1.7 1 'versicolor']
[5.7 2.5 5.0 2.0 2 'virginica']
[6.0 2.2 5.0 1.5 2 'virginica']
[6.3 2.5 5.0 1.9 2 'virginica']
[6.0 2.7 5.1 1.6 1 'versicolor']
```

Figure 1: Petal Length

```
[6.1 2.6 5.6 1.4 2 'virginica']
[6.4 3.2 4.5 1.5 1 'versicolor']
[6.9 3.1 4.9 1.5 1 'versicolor']
[6.5 2.8 4.6 1.5 1 'versicolor']
[5.9 3.0 4.2 1.5 1 'versicolor']
[5.6 3.0 4.5 1.5 1 'versicolor']
[6.2 2.2 4.5 1.5 1 'versicolor']
[6.3 2.5 4.9 1.5 1 'versicolor']
[6.0 2.9 4.5 1.5 1 'versicolor']
[5.4 3.0 4.5 1.5 1 'versicolor']
[6.7 3.1 4.7 1.5 1 'versicolor']
[6.0 2.2 5.0 1.5 2 'virginica']
[6.3 2.8 5.1 1.5 2 'virginica']
[6.3 3.3 4.7 1.6 1 'versicolor']
[6.0 2.7 5.1 1.6 1 'versicolor']
[6.0 3.4 4.5 1.6 1 'versicolor']
[7.2 3.0 5.8 1.6 2 'virginica']
[6.7 3.0 5.0 1.7 1 'versicolor']
[4.9 2.5 4.5 1.7 2 'virginica']
[5.9 3.2 4.8 1.8 1 'versicolor']
```

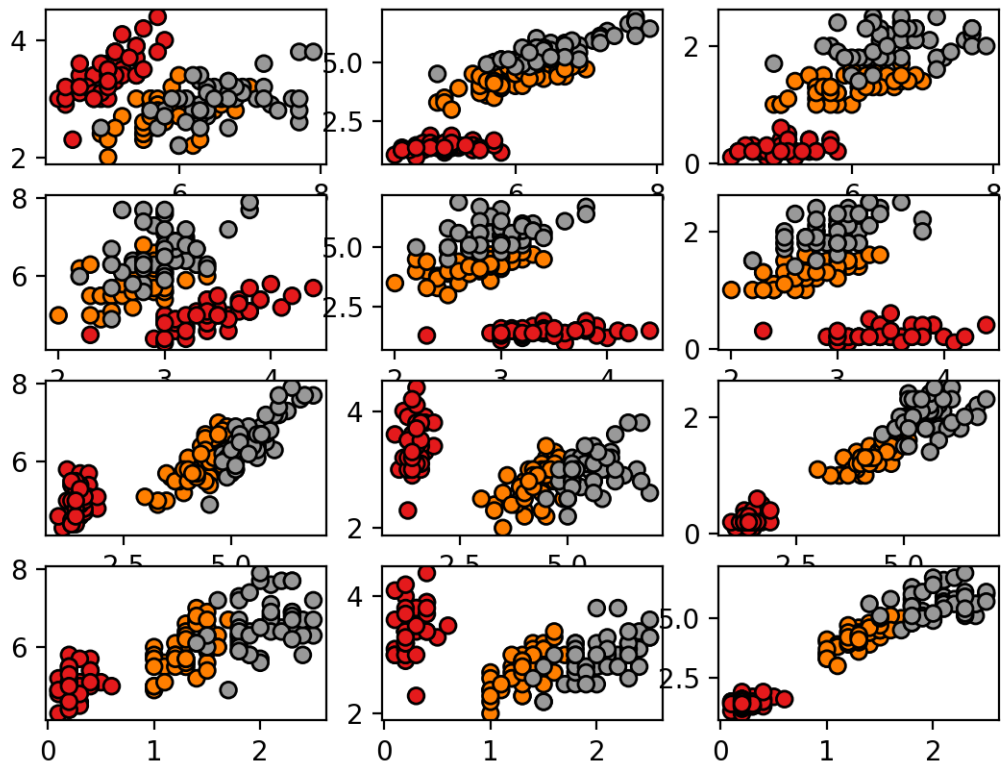
Figure 2: Petal Width

My sorting algorithm is a simple insertion sort that runs in $O(n^2)$ time. The outer loop iterates through the entirety of the dataset, by column. For each column, the sort is performed. Insertion sort will take maximum time $O(n^2)$ if sorted in reverse order, and minimum time $O(n)$ if already sorted. It's a stable, in place sort that I figured would be a good fit for the Iris dataset. It must be noted that for larger datasets, insertion sort would not be primary choice. Insertion sort is suited for a dataset of this size.

```
def insertion_sort(array, col_key, f):

    for i in range (1, len(array)):

        key = array[i, col_key]
        j   = i - 1

        while j >= 0 and key < array[j, col_key]:
            array = swap(array, j, j + 1)
            j -= 1

        array[j + 1, col_key] = key

    f.write(str(array))
    f.write("\n")
```

4. Outliers

   In consideration of outliers, one can see that there are some of them that one can see visually when features are plotted against one another.



   To take care of some outliers, I decided to remove the upper and lower 5% of the dataset, inclusive. The algorithm for outlier removal is included in `util.py`. I felt that this was an ample outlier exclusion technique considering that there are some values very close to the mean in the petal width and petal length features. Below, one can see that removal of these outliers did not greatly affect the mean or the distribution of the data. The algorithm applies a lambda function to filter for the selected quantile, which runs in $O(kn)$ time as we are making comparisons in our lambda function over a number of columns k.

```
Before removing outliers
Mean
sepal-length    5.843333
sepal-width     3.057333
petal-length    3.758000
petal-width     1.199333
dtype: float64
Skew
sepal-length    0.314911
sepal-width     0.318966
petal-length   -0.274884
petal-width    -0.102967
dtype: float64
```

Figure 3: Petal Length

```
After removing outliers
Mean
sepal-length    5.816814
sepal-width     3.039823
petal-length    3.784071
petal-width     1.208850
dtype: float64
Skew
sepal-length    0.056862
sepal-width     0.241242
petal-length   -0.516059
petal-width    -0.271300
dtype: float64
```

Figure 4: Petal Width

After removing outliers, we can see the two previous features plotted against each other again.

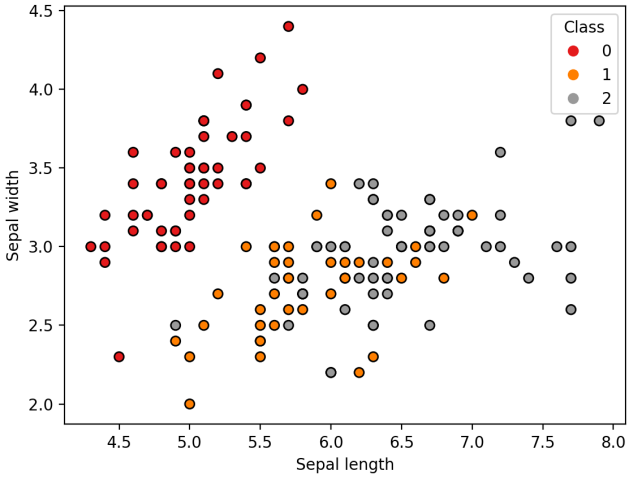


Figure 5: Before removing outliers

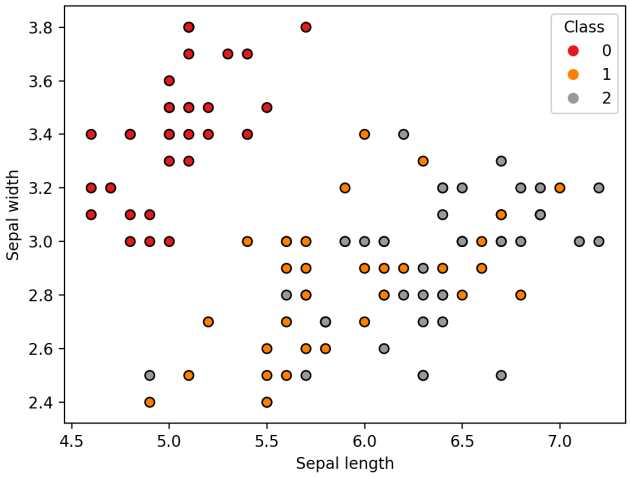


Figure 6: After removing outliers

5. Feature Ranking

   For feature ranking, I performed two different calculations of the Bhattacharyya feature ranking algorithm. The first of which draws samples from a continuous probability density function and performs a rough integration over a set number of steps, returning the Bhattacharyya coefficient. From the Bhattacharyya coefficient, we can take the log and return the negative value to get the Bhattacharyya distance. I refer to this in `iris.py` as `bhatta_cont`. The other method of calculating the Bhattacharyya distance is from the theoretical formula, referred to in `iris.py` as `bhatta_theo`. The theoretical formula is below.

$$D_B(p,q) = \frac{1}{4}\ln(\frac{1}{4}(\frac{\sigma_p^2}{\sigma_q^2} + \frac{\sigma_q^2}{\sigma_p^2} + 2)) + \frac{1}{4}(\frac{(\mu_p - \mu_q)^2}{\sigma_p^2 + \sigma_q^2})$$

   The results for both calculations is below:

```
Method: continuous density function
Feature ranking:
Bhattacharyya distance for: petal-length 8.719109725047202
Bhattacharyya distance for: petal-width 6.4662973250349385
Bhattacharyya distance for: sepal-length 0.687157942073356
Bhattacharyya distance for: sepal-width 0.5956973824662926
```

Figure 7: Bhattacharyya Distance (Continuous)

```
Method: theoretical calculation
Feature ranking:
Bhattacharyya distance for: petal-length 8.847626447883725
Bhattacharyya distance for: petal-width 6.586265223852376
Bhattacharyya distance for: sepal-length 0.8049138836932981
Bhattacharyya distance for: sepal-width 0.6892204311479477
```

Figure 8: Bhattacharyya Distance (Theoretical)

The time complexity of the continuous version of the Bhattacharyya distance algorithm should run in $O(kn)$ time, as the `bhatta_cont` function depends on the specified number of steps before running the for loop. The theoretical Bhattacharyya calculation is nearly the same, however it just runs in constant time, or $O(1)$. There are no loops or method calls, only assignment statements. From the Bhattacharyya distance, we were able to surmise an appropriate feature ranking. However, I would say that in exploring other algorithms and outlier removal methods, one could propose that there is not a way to guarantee appropriate feature ranking every single time.