Lab 6: KNN

Submitted By: Julian Singkham
Date: 01/29/2021

Abstract

The purpose of this lab is to familiarize outselves with using numpy, SKlearn, and Scipy libraries to create two variants of the KNN algorithm.

- The first variant utilizes looping to index and predict the given training/testing data.
- The second variant utilizes numpy to vectorize the training/predicting process.
- Then the two variants were compared to understand the speed advantages of vectorization.

In [1]:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from knn import KNN
from test_knn import TestKNN
import matplotlib.pyplot as plt
```

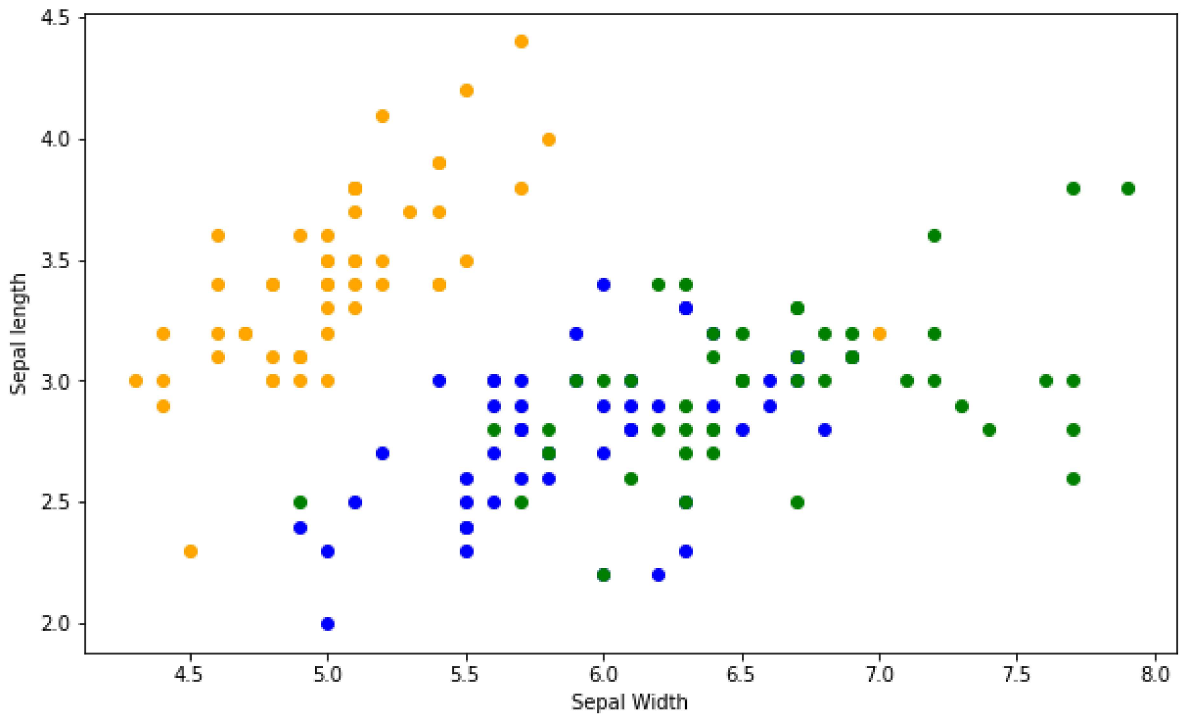
This cell plots a graph of the of sepal length vs sepal width and color codes the points by their type. The data used is the irus_dataset provided by sklearn.

In [2]:

```
iris_dataset = load_iris()
plt.figure(figsize=(10,6))
plt.plot(iris_dataset.data[:51,0], iris_dataset.data[:51,1], 'o', color = 'orange') #Setosa
plt.plot(iris_dataset.data[51:101,0], iris_dataset.data[51:101,1], 'o', color = 'blue') #Ve
plt.plot(iris_dataset.data[101:,0], iris_dataset.data[101:,1], 'o', color = 'green') #Virgi
plt.xlabel("Sepal Width")
plt.ylabel("Sepal length")
```

Out[2]:

```
Text(0, 0.5, 'Sepal length')
```



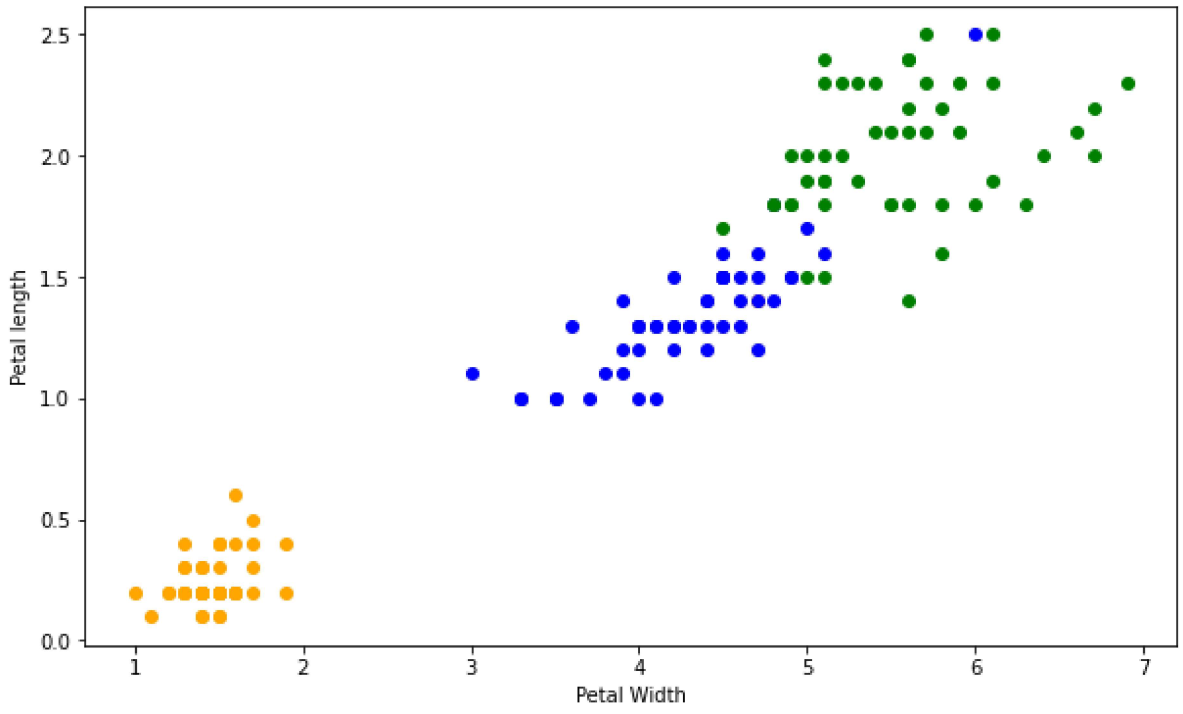
This cell plots a graph of the of petal length vs petal width and color codes the points by their type.

In [3]:

```
iris_dataset = load_iris()
iris_dataset.data[:,2:3]
plt.figure(figsize=(10,6))
plt.plot(iris_dataset.data[:51,2], iris_dataset.data[:51,3], 'o', color = 'orange') #Setosa
plt.plot(iris_dataset.data[51:101,2], iris_dataset.data[51:101,3], 'o', color = 'blue') #Ve
plt.plot(iris_dataset.data[101:,2], iris_dataset.data[101:,3], 'o', color = 'green') #Virgi
plt.xlabel("Petal Width")
plt.ylabel("Petal length")
```

Out[3]:

```
Text(0, 0.5, 'Petal length')
```



In [4]:

```
print(iris_dataset.data)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
```

Test_KNN

This section deals with running the tests provided in test_knn.py.
The implementation of KNN successfully passed all the tests. Originally the test would fail with an accuracy of 92% in iris classification. This was due to the distance calculation favoring the earlier points. This was solved by expanding the K value if the first nearest distances were exactly the same. See knn.py documentation for more information

In [5]:

```
#This section runs the provided tests
test = TestKNN()
test.test_blob_classification_loop()
test.test_blob_classification_numpy()
```

In [6]:

```
test.test_iris_classification_loop()
```

In [7]:

```
test.test_iris_classification_numpy()
```

Custom Testing of KNN

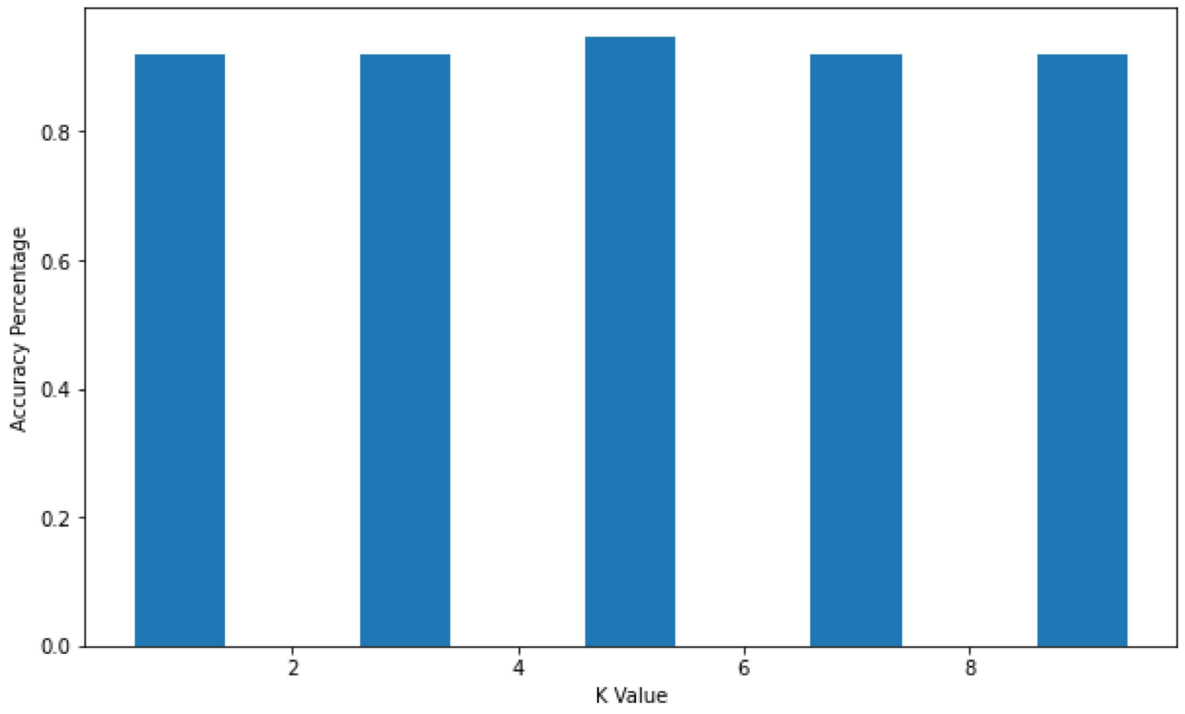
This section deals with testing the implemented KNN algorithm to find the best value for K. The first step is splitting the iris dataset into a training and testing set. The data must be split inorder to prevent over fitting of the data due to the same data set being used for training and testing. The second step is running the KNN algorithm in a loop (odd values of K) until K=11. Based on the graph, the implemented KNN algorithm has a very high accuracy at ~95% for K = 1 and 3. For K > 3 the accuracy appears to be 100%. This is most likely due to the training data not having enough information to properly classify the data. For example the training data could be composed of 50% type A, 30% type B and 20% type C. This could lead to false classifications as type A data dominates the training data and is unable to find the minute differences in the data to properly classify the data points.

In [8]:

```
train_X, test_X, train_y, test_y = train_test_split(iris_dataset.data, iris_dataset.target,
accuracy = []
k = []
for k_val in range(1,11):
    if k_val % 2 == 1:
        knn = KNN(k_val)
        knn.fit(train_X, train_y)
        accuracy.append(accuracy_score(test_y, knn.predict_numpy(test_X)))
        k.append(k_val)
plt.figure(figsize=(10,6))
plt.bar(k, accuracy)
plt.xlabel("K Value")
plt.ylabel("Accuracy Percentage")
```

Out[8]:

```
Text(0, 0.5, 'Accuracy Percentage')
```



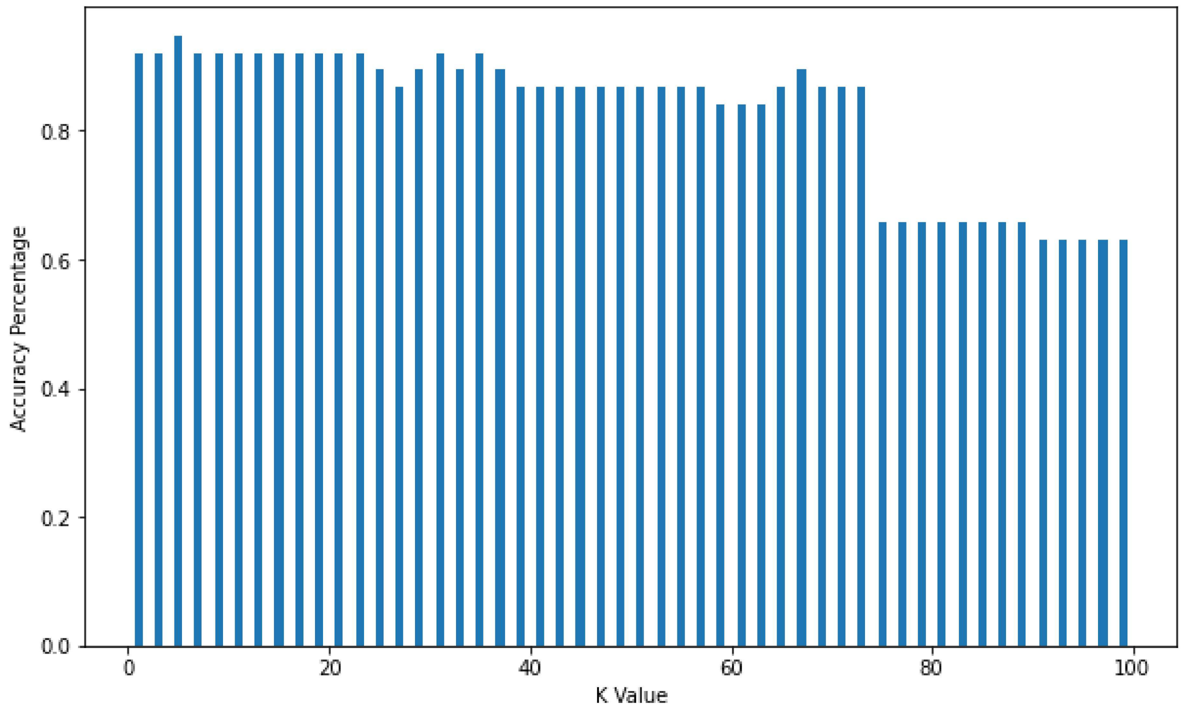
This cell deals with answering quesiton 2 in the conclusion.

In [13]:

```
accuracy = []
k = []
for k_val in range(1,100):
    if k_val % 2 == 1:
        knn = KNN(k_val)
        knn.fit(train_X, train_y)
        accuracy.append(accuracy_score(test_y, knn.predict_numpy(test_X)))
        k.append(k_val)
plt.figure(figsize=(10,6))
plt.bar(k, accuracy)
plt.xlabel("K Value")
plt.ylabel("Accuracy Percentage")
```

Out[13]:

```
Text(0, 0.5, 'Accuracy Percentage')
```



Benchmarking

This section deals benchmarking the loop variant of the KNN algorithm and the numpy variant. The testing/training data used is the data used for the "Custom Testing of KNN" section.

In [10]:

```
knn = KNN(15)
knn.fit(train_X, train_y)
```

In [11]:

```
%%timeit
knn.predict_loop(test_X)
```

```
26.3 ms ± 627 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

In [12]:

```
%%timeit
knn.predict_numpy(test_X)
```

```
1.24 ms ± 104 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

The looping variant took 25.6 ms and the numpy variant took 1.06ms to process K = 15. This difference of 25x shows how superior vectorization is compared to looping. The reason the loop variant runs 25x slowerly than the numpy variant is due to the much larger time complexity of O(# query points * # reference points^2). Since numpy utilizes vectorization it is difficult to analyze the exact O() as numpy doesn't have an easily readable concrete timing system.

Conclusion

1.) Based on the scatter plots of the features, which features (or combinations of features) appear to be able to best separate the classes?
Based off the Sepal Length vs Sepal Width and Petal Length vs Petal width, the petal width appears to be the primary difference between the flower classes. Setosa is within the range 1-2, Versicolor 3-5, and Virginica 5-7. Sepal values are too intertwined to easily differentiate class as evident with Versicolor and Virginica being mixed together.

2.)Which value of k gives the best (highest) accuracy, recall, and/or precision?
Based off the expanded Accuracy vs K Value graph, there isn't a significant drop in accuracy until k > 45 and again at k > 75. The highest level of accuracy can be found when K is within 33-43.

3.)What do you think the potential downsides of the k-nearest neighbors algorithm are? Why do you think it might not be used as widely as other methods?
The biggest disadvantage I found with using the KNN algorithm is the high memory requirement due to storing all of it's training data. Additionally due to it requiring a copy of the training data, the KNN algorithm never truely learns what it is being taught.