Project 1: Iris classification using KNN

Introduction

Put simply, the K Nearest Neighbors algorithm, is a supervised, machine learning classification algorithm that makes its predictions on a new data point based on the classification of data points closest to it. In order to develop the algorithm, the data must be split in approximately a 30-70 ratio between the test and train datasets respectively. The algorithm will then learn how to make classifications based on the train data and you can test its accuracy by comparing the predictions for the 'test' dataset against the true values.

The dataset I will be working with in this project is the iris dataset. It is a dataset of 150 enteries of different iris plants. The plants fall into 3 species, with 50 instances of each present in the dataset. The dataset also contains 4 important features about each different entry. These are the sepal length, sepal width, petal length and petal width. Based on these features, I will build my KNN classifier.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

In [2]:

```
iris_df = sns.load_dataset('iris')
```

In [67]:

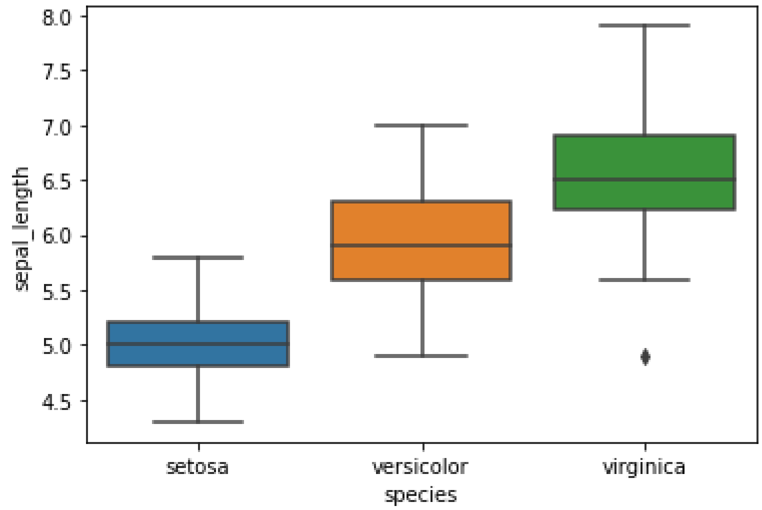
```
iris_df.head()
```

Out[67]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [64]:
```
sns.boxplot(x = 'species', y = 'sepal_length', data = iris_df)
```

Out[64]:
```
<AxesSubplot:xlabel='species', ylabel='sepal_length'>
```

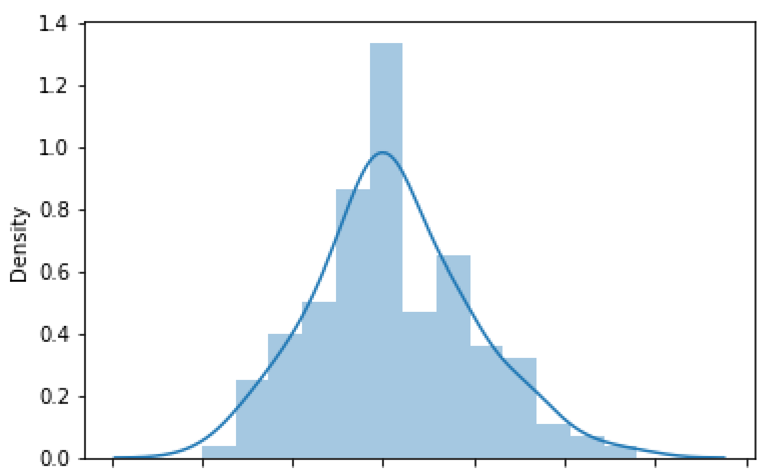



```
It appears that, on average, the virginica species has the highest average
sepal length.
```

In [70]:
```
sns.distplot(iris_df['sepal_width'])
```

```
e-level function with similar flexibility) or `histplot` (an axes-level f
unction for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[70]:
```
<AxesSubplot:xlabel='sepal_width', ylabel='Density'>
```



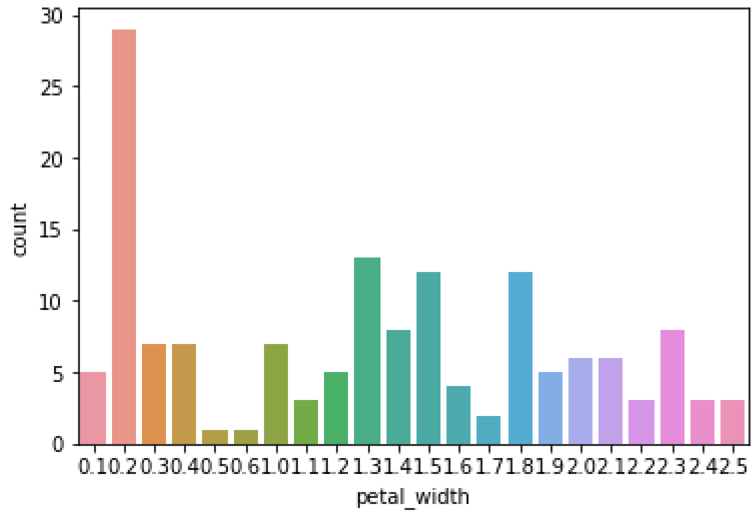
This tells us that the majority of flowers have a sepal width concentrated around 3.0mm

In [75]:

```
sns.countplot(x='petal_width', data = iris_df)
```

Out[75]:

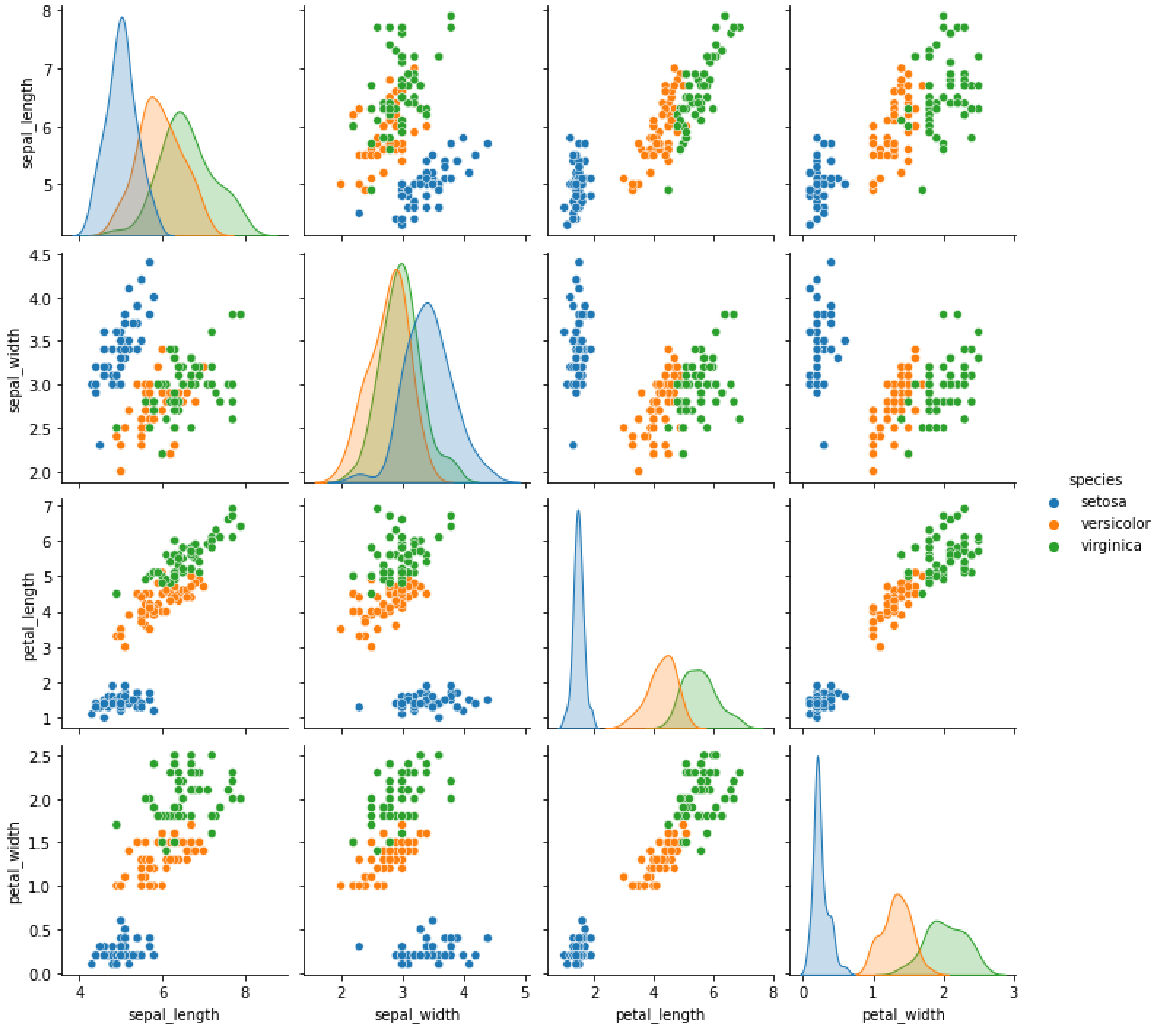
```
<AxesSubplot:xlabel='petal_width', ylabel='count'>
```



This shows us the count of each unique value in the petal_width column. This helps us see what the most common petal_width is.

In [76]:
```
sns.pairplot(iris_df, hue = 'species')
```

Out[76]: <seaborn.axisgrid.PairGrid at 0x15a2cf9c9a0>



This pairplot allows us to view the relationship between different features in our dataset, based on species type.

Classifier with 4 features

In [4]:
```
scaler = StandardScaler()
scaler.fit(iris_df.drop('species', axis = 1))
```

Out[4]: StandardScaler()

In [5]:
```
scaled_four_features = scaler.transform(iris_df.drop('species', axis = 1))
```

In [7]:
```
iris_df_four_features = pd.DataFrame(scaled_four_features, columns = iris_df.
```

In [8]:
```
iris_df_four_features.head()
```

Out[8]:

	sepal_length	sepal_width	petal_length	petal_width
0	-0.900681	1.019004	-1.340227	-1.315444
1	-1.143017	-0.131979	-1.340227	-1.315444
2	-1.385353	0.328414	-1.397064	-1.315444
3	-1.506521	0.098217	-1.283389	-1.315444
4	-1.021849	1.249201	-1.340227	-1.315444

In [9]:
```
x = iris_df_four_features
y = iris_df['species']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, rar
```

In [41]:
```
knn = KNeighborsClassifier(n_neighbors = 2)
knn.fit(x_train, y_train)
prediction_one = knn.predict(x_test)
```

In [42]:
```
prediction_one
```

Out[42]:
```
array(['setosa', 'setosa', 'setosa', 'virginica', 'versicolor',
       'virginica', 'versicolor', 'versicolor', 'versicolor', 'setosa',
       'virginica', 'setosa', 'setosa', 'virginica', 'virginica',
       'versicolor', 'versicolor', 'versicolor', 'setosa', 'versicolor',
       'versicolor', 'setosa', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'virginica', 'setosa', 'setosa',
       'virginica', 'versicolor', 'virginica', 'versicolor', 'virginica',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'virginica', 'setosa', 'setosa', 'setosa', 'versicolor',
       'versicolor', 'setosa', 'virginica', 'versicolor', 'setosa',
       'versicolor'], dtype=object)
```

In [43]:
```
print (confusion_matrix(y_test, prediction_one))
print(classification_report(y_test, prediction_one))
```

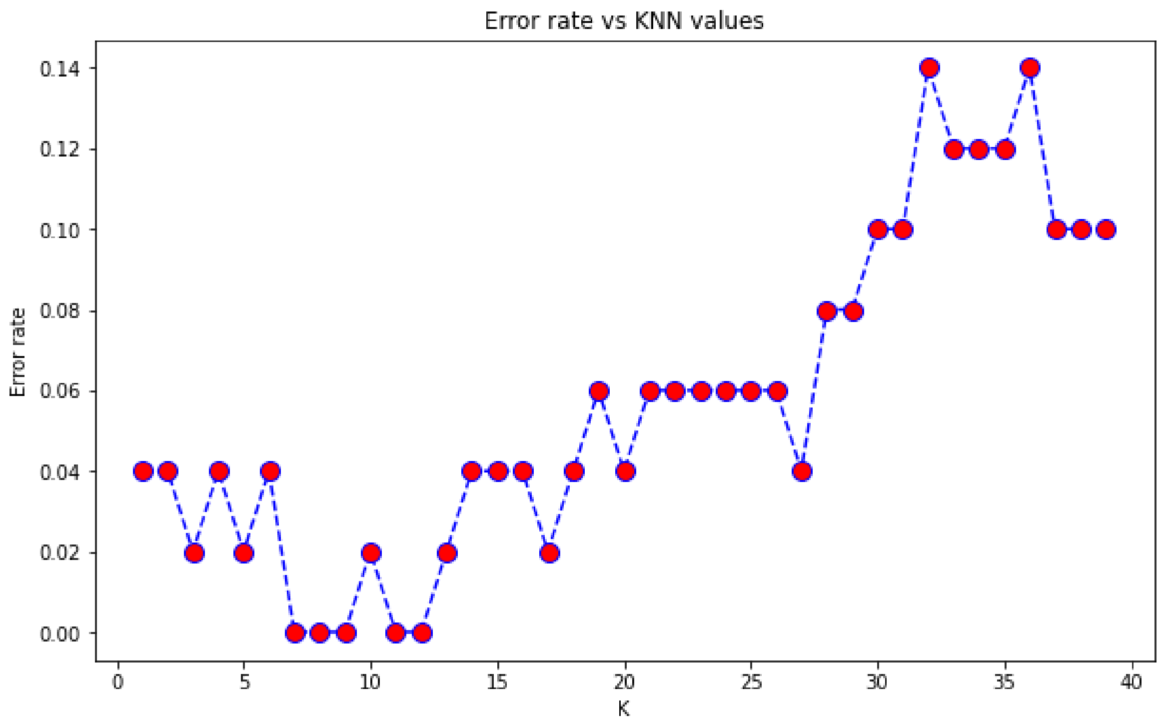
```
[[15  0  0]
 [ 0 22  0]
 [ 0  2 11]]
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       0.92      1.00      0.96        22
   virginica       1.00      0.85      0.92        13

    accuracy                           0.96        50
   macro avg       0.97      0.95      0.96        50
weighted avg       0.96      0.96      0.96        50
```

This table tells us that when using 2 neighbors for our classifier, we are able to correctly identify members of the 'setosa' species correctly every time. Meanwhile our classifications for the 'versicolor' and 'virginica' are good, but not 100% accurate. We will try to see if there are any better values for k that will yield better results.

In [27]:
```
error_rate = []
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors = i)
    knn.fit(x_train, y_train)
    prediction_i = knn.predict(x_test)
    error_rate.append(np.mean(prediction_i != y_test))
```

In [28]:
```
plt.figure(figsize = (10,6))
plt.plot(range(1,40), error_rate, color = 'blue', linestyle = 'dashed', marke
plt.title('Error rate vs KNN values')
plt.xlabel('K')
plt.ylabel('Error rate')
```

Out[28]: Text(0, 0.5, 'Error rate')



From our plot, we can see that the best number of neighbors to use is around 7-9. Using this value in our classifier will give us the results:

In [44]:
```
knn = KNeighborsClassifier(n_neighbors = 7)
knn.fit(x_train, y_train)
prediction_one = knn.predict(x_test)
print (confusion_matrix(y_test, prediction_one))
print(classification_report(y_test, prediction_one))
```

```
[[15  0  0]
 [ 0 22  0]
 [ 0  0 13]]
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        15
  versicolor       1.00      1.00      1.00        22
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        50
   macro avg       1.00      1.00      1.00        50
weighted avg       1.00      1.00      1.00        50
```

Here we can see that when we used k = 7, our results were much more accurate, with our classifier correctly identifying every element in the dataset.

Classifier with 2 features

In [45]:
```
iris_df.head()
```

Out[45]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [48]:
```
scaler_two = StandardScaler()
scaler_two.fit(iris_df.drop(columns = ['petal_length', 'petal_width', 'specie
```

Out[48]: StandardScaler()

In [50]:
```
scaled_two_features = scaler_two.transform(iris_df.drop(columns = ['petal_ler
```

In [51]:
```
iris_df_two_features = pd.DataFrame(scaled_two_features, columns = iris_df.cc
```

In [52]:
```
iris_df_two_features.head()
```

Out[52]:

	sepal_length	sepal_width
0	-0.900681	1.019004
1	-1.143017	-0.131979
2	-1.385353	0.328414
3	-1.506521	0.098217
4	-1.021849	1.249201

In [53]:
```
x = iris_df_two_features
y = iris_df['species']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, rar
```

In [55]:
```
knn_two = KNeighborsClassifier(n_neighbors = 2)
knn_two.fit(x_train,y_train)
prediction_two = knn_two.predict(x_test)
```

In [56]:
```
print (confusion_matrix(y_test, prediction_two))
print (classification_report(y_test, prediction_two))
```

```
[[14  1  0]
 [ 0 14  8]
 [ 0 10  3]]
              precision    recall  f1-score   support

      setosa       1.00      0.93      0.97        15
  versicolor       0.56      0.64      0.60        22
   virginica       0.27      0.23      0.25        13

    accuracy                           0.62        50
   macro avg       0.61      0.60      0.60        50
weighted avg       0.62      0.62      0.62        50
```

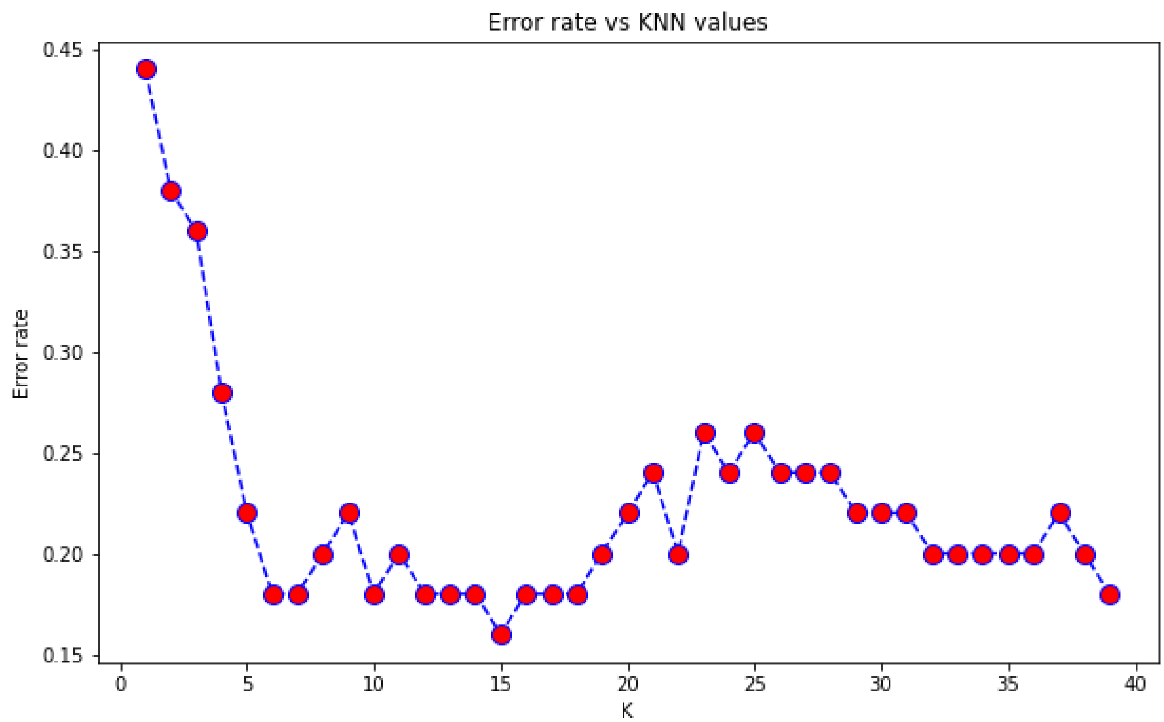
This information tells us that our current classifier with k = 2, is not the best. It is able to correctly classify members of the 'setosa' species with great accuracy but the other species are not nearly as accurate. We will plot the error rate of different values of k to find the best one to use.

In [57]:
```
error_rate_two = []
for i in range(1,40):
    knn_two = KNeighborsClassifier(n_neighbors = i)
    knn_two.fit(x_train, y_train)
    prediction_i = knn_two.predict(x_test)
    error_rate_two.append(np.mean(prediction_i != y_test))
```

In [58]:

```
plt.figure(figsize = (10,6))
plt.plot(range(1,40), error_rate_two, color = 'blue', linestyle = 'dashed', m
plt.title('Error rate vs KNN values')
plt.xlabel('K')
plt.ylabel('Error rate')
```

Out[58]: Text(0, 0.5, 'Error rate')



According to the graph, when we set k to equal 15, we should have our lowest error rate.

In [62]:

```
knn_two = KNeighborsClassifier(n_neighbors = 15)
knn_two.fit(x_train,y_train)
prediction_two = knn_two.predict(x_test)
```

In [63]:
```
print (confusion_matrix(y_test, prediction_two))
print (classification_report(y_test, prediction_two))
```

```
[[14  1  0]
 [ 0 17  5]
 [ 0  2 11]]
              precision    recall  f1-score   support

      setosa       1.00      0.93      0.97        15
  versicolor       0.85      0.77      0.81        22
   virginica       0.69      0.85      0.76        13

    accuracy                           0.84        50
   macro avg       0.85      0.85      0.84        50
weighted avg       0.85      0.84      0.84        50
```

As we can see based on the f1-score, when we used k = 15, our classifier was quite significantly more accurate for the second and third species, while still keeping the initial accuracy for the first species.

Comparing our two classifiers, we can easily see that the first one we created was a much better predictor. This is because in our first classifier, we used all 4 features in making our predictions, which gave our algorithm more opportunities to learn based on different information and therefore produced a more accurate algorithm. In the second classifier, we only used two features, and therefore ignored some crucial data that determines the classification for each entry in the dataset.

Conclusion

This project showed that it is important to consider all, or at least all of the relevant, features when designing the knn algorithm in order to make your classifier as accurate as possible. We can judge how well a classifier is performing by analyzing the confusion matrix and the classification report between our y_test and prediction columns. After that we have to determine the best value of k to use by plotting the error rates for various values of k.