Machine Learning: The Iris Problem

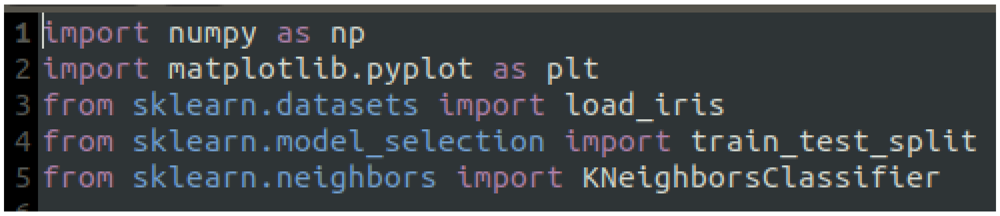
Introduction:

Machine learning is a large new field in computational search and development. One of the classic problems that kickstarted this field of learning computer software is the Iris Flower Data set. This tutorial utilizes Python to gather the iris data set, analyze some key features, and then determine a decision based off Python's integrated sklearn library for what kind of iris flower is presented by key features. For more information on the features of the iris data set please visit the link below.

Iris Data set: https://en.wikipedia.org/wiki/Iris_flower_data_set

Procedure:

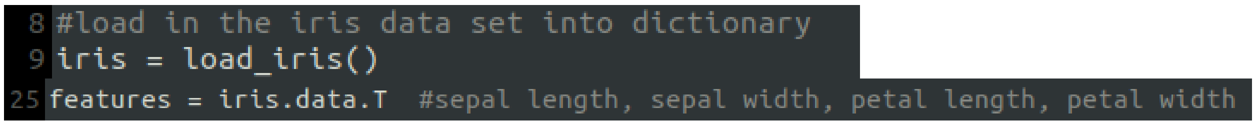
- Using either Python or Python3 begin a new .py file and import the libraries listed in figure 1 below. These will be used throughout the program to analyze the data set.
  - numpy: fundamental scientific computing (using its arrays functionality)
  - matplotlib.pyplot: will be used for graphing the data
  - sklearn
    - datasets: importing the iris data into our program
    - model_selection: splits arrays into subsets and tests based on parameters
    - neighbors: using KNN groups nearby data into clusters



```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

Figure 1: necessary Python libraries

- Access the features of the data set after loading into a new dictionary variable to store all the data. These will then be transposed using the .T operation to make the data set easier to deal with. *Note there is some commented out code through the source file listed on the website; this commented out code is useful to help visualize what is happening, uncomment it as you choose.



```python
#load in the iris data set into dictionary
iris = load_iris()
features = iris.data.T  #sepal length, sepal width, petal length, petal width
```

Figure 2: loading data set

- After loading the data set please study its features. A feature is an element that can be described or studied to help us classify an object. In this case the features of the data set are the recorded measurements for sepal length and width and petal length and width.

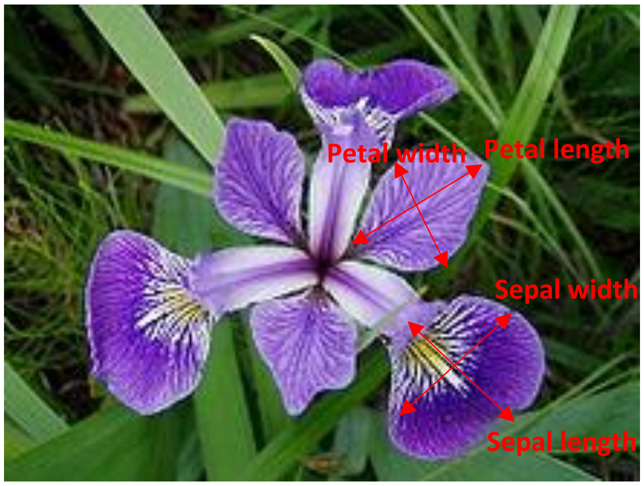


Figure 3: Iris features (Versicolor)

- The other element of the data set we have yet to talk about is the classification. We want to classify the type of Iris flower, either a versicolor, virginica, or setosa, based on its features automatically. Therefor the last array in the matrix is reserved for the type of Iris the preceding data describes.

- After parsing our data into several key features, we can start finding correlations between the features and the type of Iris it is. To begin we will start by getting a visualization of what is occurring by plotting some of the features against one another and looking for a clustering correlation. Two of many plots possible to be made are shown below. The data points in the plot are colored not based on their location but what classification they belong to in the data set; so, all setosas are one color or all virginica is another color. Seeing these correlations is the beginning of being able to train our model to determine which features determine which flowers.

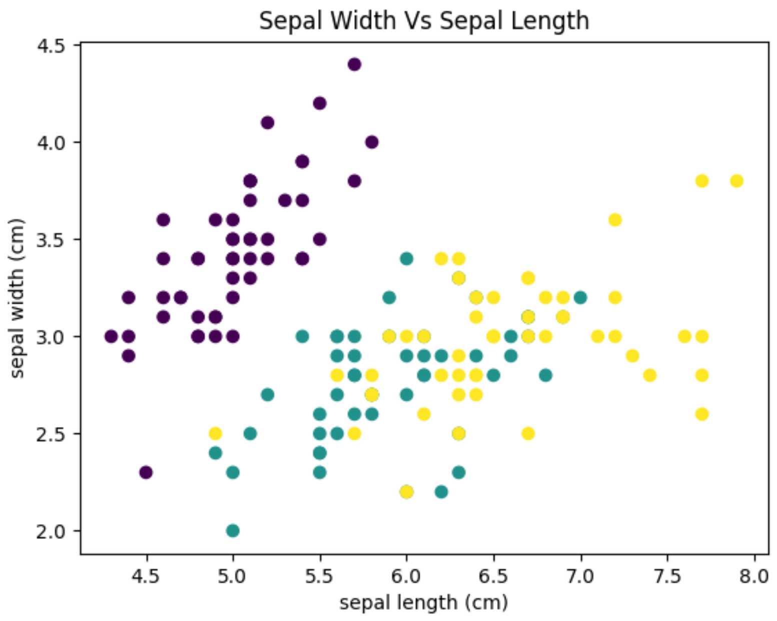


Figure 4: Sepal W and L correlations

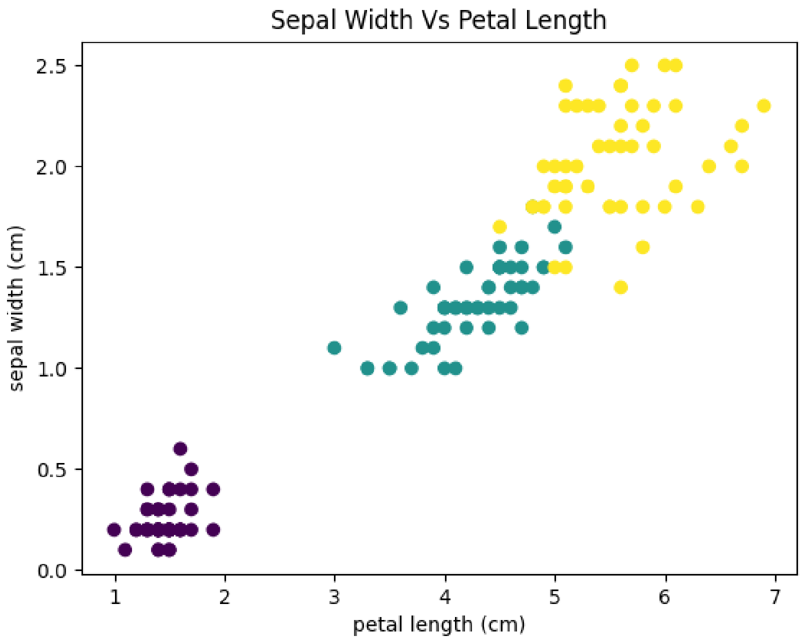


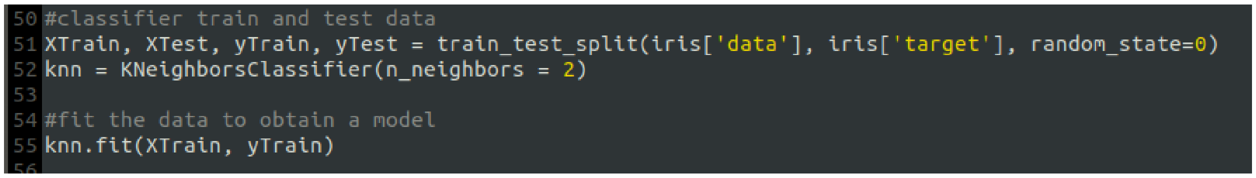
Figure 5: Sepal W and Petal L correlations

**Note the following will be using a k nearest neighbor (KNN) algorithm to sort the data above, for more information on how KNN works please check here: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

- Now that we have seen some distributions and some visual correlations we can use the train_test_split to automatically separate the data into smaller subsets that can be individual trained to match a correlation, and then be pooled back together. In other words, it is dividing the data we must first create a model and then use the remaining data to test if the model is accurate. It is given the whole data set and what we are targeting to classify. In this case we want to determine the name or type of iris flower. When a random state of 0 is passed into this function the train_test_split will automatically replace it with a 75/25 divide for 75% of the data will be used to train the model.

- After training the data we will create a KNN classifier variable to run the KNN algorithm. Since we are trying to classify only 3 types of flowers, and we only have 150 data points, we will use a k of 2 for each data point the algorithm tests. This means that it will look at the 2 closest data points to the one being looked at to add it to a current cluster.

- The KNN classifier needs to then be fit to the training data that was specified with the train_test_split function call. Fitting the data here to the classifier results in a model that can predict to a decent degree of accuracy what kind of flower is presented by the data that is fed to it.



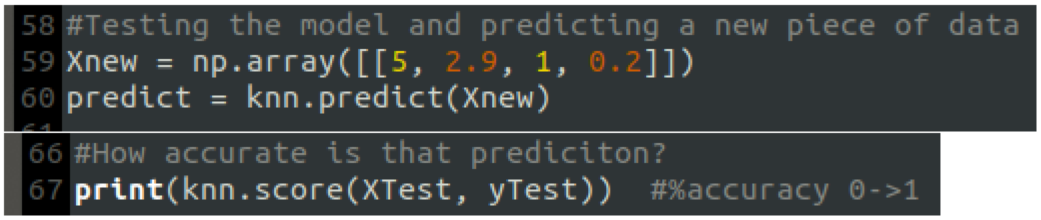
```
#classifier train and test data
XTrain, XTest, yTrain, yTest = train_test_split(iris['data'], iris['target'], random_state=0)
knn = KNeighborsClassifier(n_neighbors = 2)

#fit the data to obtain a model
knn.fit(XTrain, yTrain)
```

Figure 6: Classifier and training commands

- Now that a k-nearest neighbors model has been fit to a certain subset of the data used for training the correlations, we can test it using a data point in the set that was not used for training.

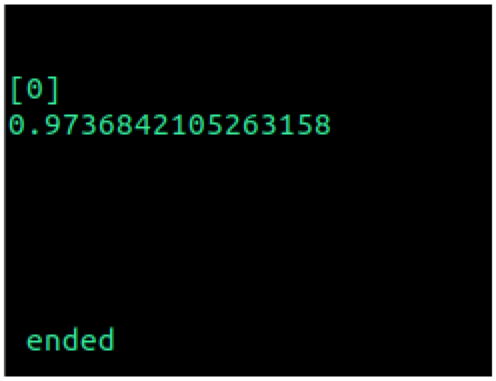
- To help better understand how well this classifier model works, we can also see what the probability is that the guess is a right guess. The lines needed and the subsequent output for the test that is run is seen below.



```
#Testing the model and predicting a new piece of data
Xnew = np.array([[5, 2.9, 1, 0.2]])
predict = knn.predict(Xnew)
```

```
#How accurate is that prediciton?
print(knn.score(XTest, yTest))  #%accuracy 0->1
```

Figure 7: prediction



```
[0]
0.9736842105263158




 ended
```

Figure 8: Result for predict (setosa 97% confidence)