Iris Dataset

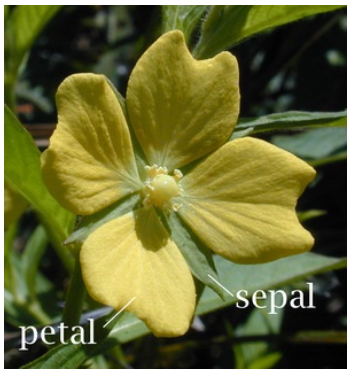
The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant.

Iris Setosa

Iris Versicolor

Iris Virginica

Each instance has four measurements from the flowers (sepal length, sepal width, petal length, and petal width).



Pre-processing

To import the dataset execute the following code:

```
from sklearn.datasets import load_iris

iris = load_iris()
iris_data = iris.data
iris_label = iris.target
# print the features names and class names
print(iris.feature_names)
print(iris.target_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
['setosa' 'versicolor' 'virginica']
```

To create training and test splits, execute the following script:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(iris_data, iris_label, test_size=0.20)
```

The above script splits the dataset into 80% train data and 20% test data. This means that out of total 150 records, the training set will contain 120 records and the test set contains 30 of those records.

Visualization

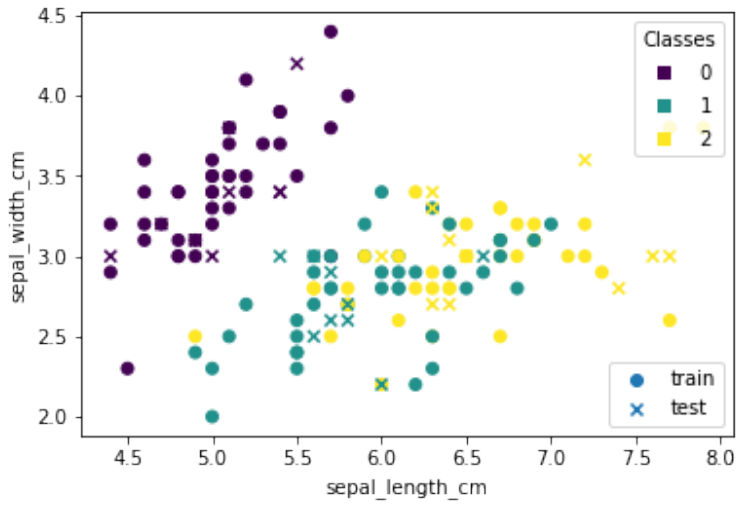
To draw the scatter plot of the first two features for training and testing data :

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
train = ax.scatter(X_train[:,0], X_train[:,1], c=y_train,marker='o')
test = ax.scatter(X_test[:,0], X_test[:,1], c=y_test, marker='x')

# produce a legend with the unique colors from the scatter
legend1 = ax.legend(*train.legend_elements(), title="Classes",loc="upper right")
ax.add_artist(legend1)

# produce a legend for different marks
legend2 = ax.legend((train, test),('train', 'test'), loc="lower right")
plt.xlabel('sepal_length_cm')
plt.ylabel('sepal_width_cm')
plt.show()
```



Training and Predictions

It is extremely straight forward to train the KNN algorithm and make predictions with it, especially when using Scikit-Learn.

To train a KNN classifier with K=5 (5 neighbors):

```
# import the KNeighborsClassifier class from the sklearn.neighbors library.
from sklearn.neighbors import KNeighborsClassifier
# initialize the class with 5 neighbors to use.
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
```

To make predictions on test data, execute the following script:

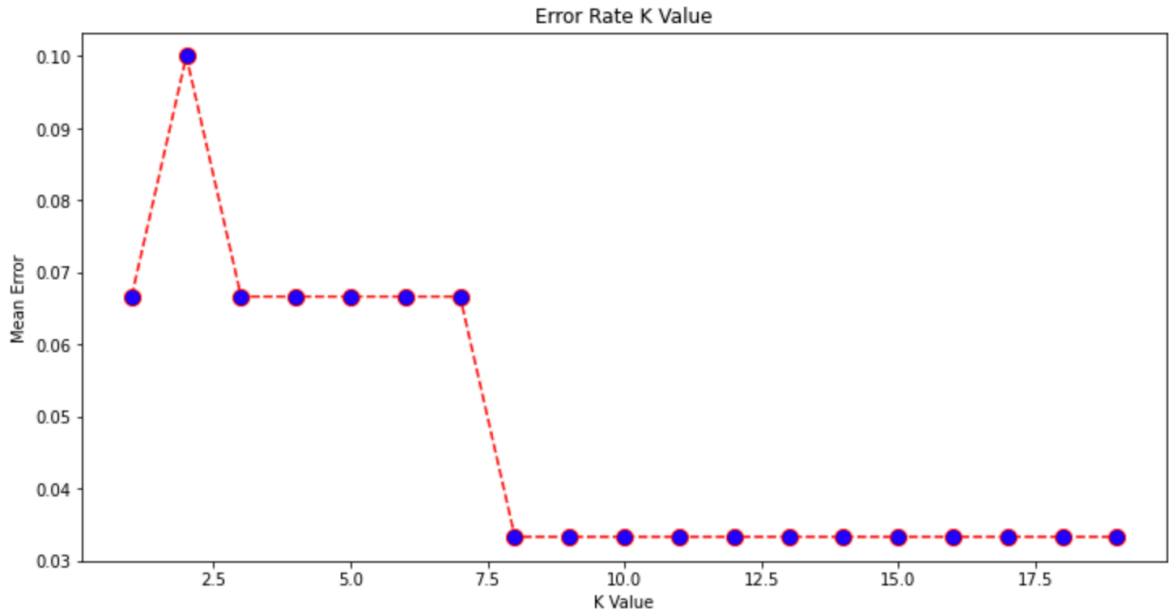
```
y_pred = classifier.predict(X_test)
```

Try different K and report results

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
# Calculating error for K values between 1 and 20
error = []
for i in range(1, 20):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(1-accuracy_score(pred_i, y_test))
```

To plot the error values against K values, execute the following script to create the plot:

```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 20), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
plt.show()
```



To use Manhattan distance function, initialize the classifier with Manhattan distance metric:

```
knn = KNeighborsClassifier(n_neighbors=10,metric='manhattan')
```

The distance metrics you can use are listed in the following table

"euclidean"	EuclideanDistance	•	`sqrt(sum((x - y)^2))`
"manhattan"	ManhattanDistance	•	`sum(|x - y|)`
"chebyshev"	ChebyshevDistance	•	`max(|x - y|)`
"minkowski"	MinkowskiDistance	p	`sum(|x - y|^p)^(1/p)`
"wminkowski"	WMinkowskiDistance	p, w	`sum(|w * (x - y)|^p)^(1/p)`
"seuclidean"	SEuclideanDistance	V	`sqrt(sum((x - y)^2 / V))`
"mahalanobis"	MahalanobisDistance	V or VI	`sqrt((x - y)' V^-1 (x - y))`

To apply z-normalization to training and testing data, use the following script:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

To apply revised voting scheme, initialize the classifier with self-defined weight function corresponding to the scheme:

```
#define the weight
def my_distance(distance):
   return 1/(distance**2+1e-20)

knn = KNeighborsClassifier(n_neighbors=10, weights=my_distance)
```

The possible values for parameter weight are listed as follows:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

You can try other datasets provided in sklearn.datasets given in the following link https://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets. For example, to use the breast cancer wisconsin dataset, use the following script:

```
from sklearn.datasets import load_breast_cancer
X, y = load_breast_cancer(return_X_y=True)
print(X.shape)
print(y.shape)
```

```
(569, 30)
(569,)
```