In [1]:
```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
data = pd.read_csv("iris.csv",sep=',')
```

In [3]:
```
Y = data['class']
X = data.drop(['class'],axis=1)
```

In [7]:
```
def train_test_split(X,Y,test_size=0.25, random=False, random_seed=None):

    X = np.array(X)
    Y = np.array(Y)

    indices = np.array(range(len(X)))

    test_size_len = round(test_size * X.shape[0])

    if random == True:
        if random_seed != None:
            random_generator = np.random.RandomState(seed=random_seed)
            random_generator.shuffle(indices)

        else:
            np.random.shuffle(indices)


        test_indices = indices[0:test_size_len]
        train_indices = indices[test_size_len:]

        X_train = X[train_indices, : ]
        X_test = X[test_indices, : ]
        Y_train = Y[train_indices]
        Y_test = Y[test_indices]

    else:
        train_indices =  indices[0:(len(X)-test_size_len)]
        test_indices = indices[(len(X)-test_size_len):]

        X_train = X[train_indices, : ]
        X_test = X[test_indices, : ]
        Y_train = Y[train_indices]
        Y_test = Y[train_indices]

    return X_train,X_test,Y_train,Y_test
```

In [14]:
```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,random=True,test_size=0.3,random_seed=50)
```

유클라디안 거리 구하기

In [ ]:
```
def euclidean_dist(obs1,obs2):
    dist = np.sqrt(np.sum((obs1-obs2)**2))
    return round(dist,2) # 소수점 아래 2자리에서 반올림
```

맨허튼 거리 구하기

In [ ]:
```
def manhattan_dist(obs1,obs2):
    return np.sum(np.abs(obs1-obs2))
```

In [ ]:
```
def search_neighbors2(X_train,test_sample,k=5):

    dists_info = list()

    for index,train_sample in enumerate(X_train):
        dist = manhattan_dist(train_sample,test_sample)
        dists_info.append((dist,index,train_sample))

    dists_info.sort(key=lambda x : x[0])

    return dists_info[:k]
```

K이웃 탐색

In [7]:
```
def search_neighbors(X_train,test_sample,k=5):

    dists_info = list()

    for index,train_sample in enumerate(X_train):
        dist = euclidean_dist(train_sample,test_sample)

        dists_info.append((dist,train_sample,index))

    dists_info.sort(key=lambda tupe:tupe[0])
    neighbors = dists_info[1:k+1]

    return neighbors
```

K 이웃 탐색 함수를 이용한 Knn prdiction

In [8]:
```
def knn_prediction (X_train,Y_train,test_sample,k=5):

    neighbors = search_neighbors(X_train,test_sample,k)
    neigh_index = list()
    for neigh in neighbors:
        neigh_index.append(neigh[2])

    neigh_Y = Y_train[neigh_index]

    classes, counts = np.unique(neigh_Y, return_counts=True)

    pred = classes[np.argmax(counts)]

    return pred,neigh_Y
```

In [9]:
```
test_sample = X_train[100,:]
```

In [10]:
```
knn_prediction(X_train,Y_train,test_sample, 15)
```

Out[10]:
```
('Iris-versicolor',
 array(['Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
        'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
        'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
        'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
        'Iris-versicolor', 'Iris-versicolor', 'Iris-virginica'],
       dtype=object))
```

전체 데이터 셋에 대해서 예측 및 평가하기

In [11]:
```
total_pred = list()
for index, test_sample in enumerate(X_test):

    pred, neigh_y = knn_prediction(X_train, Y_train, test_sample, k=5)

    total_pred.append(pred)
```

In [12]:
```
total_accuracy = sum(total_pred==Y_test)/len(Y_test)
```

In [13]:
```
total_error_rate= 1 - total_accuracy
```

In [15]:
```
print("정확도 = {: .2f}".format((total_accuracy)*100))
print("오류율 = {: .2f}".format((total_error_rate)*100))
```

```
정확도 =  97.78
오류율 =  2.22
```

클래스 안에 넣기

In [9]:

```
class Knn:

    def __init__(self,k=5):

        self.k = k
        self.neighbors = None


    def euclidean_dist(self,obs1,obs2):

        dist = np.sqrt(np.sum(obs1-obs2)**2)

        return np.round(dist,3)

    def search_neighbors(self,X_train,test_sample):
        dists_info = list()

        for index, train_sample in enumerate(X_train):

            dist = self.euclidean_dist(train_sample, test_sample)

            dists_info.append((train_sample,dist,index))

        dists_info.sort(key=lambda tupe:tupe[1])

        self.neighbors = dists_info[:self.k]

        return self.neighbors

    def predict (self,X_train,Y_train, test_sample):

        self.neighbors = self.search_neighbors(X_train,test_sample)

        # neigh_index = [neigh[2] for neigh in self.neighbors]#

        neigh_index = list()
        for neigh in self.neighbors:
            neigh_index.append(neigh[2])

        neigh_Y = Y_train[neigh_index]

        classes, counts = np.unique(neigh_Y, return_counts=True)

        pred = classes[np.argmax(counts)]

        return pred,neigh_Y
```

In [15]:
```
knn = Knn(k=5)
```

In [16]:
```
total_pred = list()

for index,test_sample in enumerate(X_test):
    pred, neigh_y = knn.predict(X_train,Y_train, test_sample)

    total_pred.append(pred)
```

In [17]:
```
knn.neighbors
```

Out[17]:
```
[(array([6.9, 3.2, 5.7, 2.3]), 0.0, 57),
 (array([7.3, 2.9, 6.3, 1.8]), 0.2, 25),
 (array([6.7, 3.3, 5.7, 2.1]), 0.3, 58),
 (array([6.3, 3.4, 5.6, 2.4]), 0.4, 52),
 (array([7.2, 3. , 5.8, 1.6]), 0.5, 43)]
```

In [18]:
```
total_accuracy = sum(total_pred==Y_test)/len(Y_test)
total_error_rate = 1 - total_accuracy

print("정확도 = {:.2f}".format((total_accuracy)*100))
print("오류율 = {:.2f}".format((total_error_rate)*100))
```

```
정확도 = 84.44
오류율 = 15.56
```