REPORT

홍익대학교 HONGIK UNIVERSITY

- K-Nearest Neighbor -

HW1 : Iris classification

과목명	:	인공지능
담당교수	:	박 준
학년	:	4학년
학번	:	B411241
이름	:	홍성민
제출일	:	20.04.15

1. 과제 개요

K-Nearest Neighbor 알고리즘을 이용하여 Iris classification(분류) 문제 해결한다. Majority vote, Weighted Majority vote 두가지 방법을 모두 사용한다.

2. 구현 환경

언어 : python3 / tool : pycharm2019.3.4 / 운영체제 : window10

3. KNN (majority vote, weighted majority vote) 알고리즘에 대한 설명

- KNN (K-Nearest Neighbor) 알고리즘

KNN은 지도학습 중 분류 문제에 사용하는 알고리즘이며 분류라는 것은 데이터의 답이 어떤 그룹에 속하는지 판별되는 문제를 말한다. KNN은 새로 들어온 데이터 Y에 대해 가장 가까운 데이터를 찾으며 가장 가까운 데이터로 판별 된 것이 A라면 Y는 A그룹이다라고 분류하는 알고리즘이다. 거리는 유클리드 거리 계산으로 하며 이것은 feature 개수와 상관없이 동일하게 적용한다. 공식은 아래와 같다.

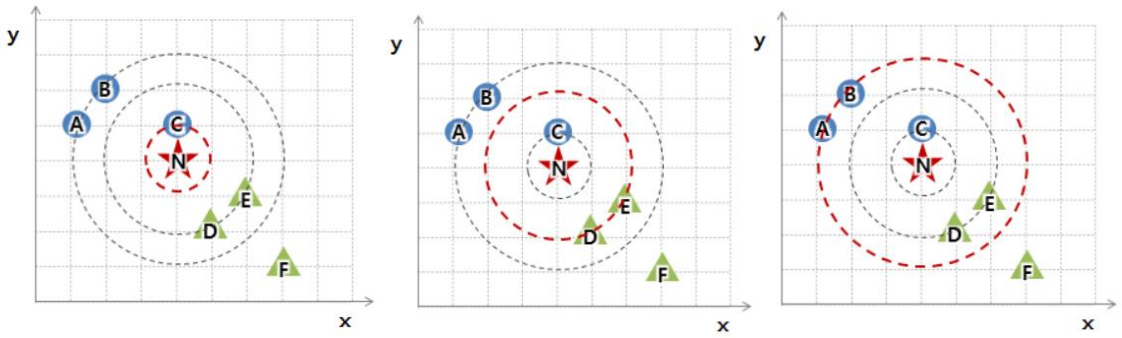
$\mathbf{p} = (p_1, p_2, \ldots, p_n)$와 $\mathbf{q} = (q_1, q_2, \ldots, q_n)$

$$\|\mathbf{p}-\mathbf{q}\| = \sqrt{(\mathbf{p}-\mathbf{q})\cdot(\mathbf{p}-\mathbf{q})} = \sqrt{\|\mathbf{p}\|^2 + \|\mathbf{q}\|^2 - 2\mathbf{p}\cdot\mathbf{q}}.$$

K-Nearest Neighbor 알고리즘에서 K라는 것은 몇 번째로 가까운 데이터까지 살펴볼 것인가를 정하는 숫자이다. K값에 따라서 알고리즘의 결과가 다르게 나오게 되며 이것은 아래에서 더 살펴보도록 하겠다.

- Majority vote

기본적인 KNN 알고리즘 방법으로 K개의 가장 가까운 데이터를 보고 그중 가장 많이 나온 그룹으로 답을 책정하는 것이다. 아래 그림은 K값에 따라서 답이 다르게 나오는 것을 보여주는 그림이다.



위 그림을 보게 되면 N은 새로 들어온 테스트 데이터를 의미하고 A,B,C는 파란색 원 그룹, D,E,F는 연두색 삼각형 그룹을 의미한다. 맨 왼쪽 그림은 K=1 일 때 상황이며 가장 가까운 데이터는 C로 N은 파란색 원 그룹으로 책정된다. 가운데 그림은 K=3 일 때 상황이며 가장 가까운 데이터는 C,D,E로 연두색 삼각형 그룹이 2개로 더 많으므로 N은 연두색 삼각형 그룹으로 책정된다. 마지막으로 오른쪽 그림은 K=5 일 때 상황으로 가까운 데이터는 A,B,C,D,E로 파란색 원 그룹이 3개로 더 많으므로 N은 파란색 원 그룹으로 책정된다. 이렇듯 K값에 따라 KNN 알고리즘의 결과는 다르게 나올 수 있다.

- Weighted Majority vote

Majority vote는 K개의 가까운 데이터의 개수로 결과를 책정했다면 Weighted Majority vote는 데이터를 거리에 가중치를 두어 계산해 주는 것을 말한다. 예를 들어 새로 들어온 데이터 N을 5개의 A그룹 데이터가 매우 가까운 거리에서 둘러싸고 있다고 상정해보자. 하지만 그 밖으로는 거리가 좀 있는 B그룹 데이터가 50개 있다면 K가 10이상일 경우 Majority vote에 의해서 N은 무조건 B로 책정되는 문제가 있다. 이것을 해결하고자 가까운 거리에 있는 데이터에 대해서는 더욱 가중치를 주어 결과를 내놓는 것이 Weighted Majority vote인 것이다. 가중치를 주는 방법에는 여러가지가 있을 수 있으나 통상적으로 쓰이는 가중치는 1/거리 이며 과제에도 이것을 도입할 것이다.

4. 데이터에 대한 설명

python의 Scikit-learn 라이브러리에서 제공하는 Iris Data를 사용한다. 데이터의 개수는 총 150개가 있다.

4.1 Input Feature

data set feature로는 ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'] 이렇게 총 4종류가 있으며 sepal은 꽃받침, petal은 꽃잎을 말한다.

4.2 Target Output

data set 답으로는 0: Setosa, 1: Versicolor, 2: Virginica 총 세개의 분류 집합이 있다. 꽃의 이름을 의미하는 output이다.

5. 소스코드 설명

5.1 KNN 클래스 (knnclass.py) 설계, 구현 및 소스코드

<설계>

(1) 클래스 생성자 인자, 클래스 변수 설계

- 클래스 생성자 인자 (k, xtrain, ytrain, yname)

우선 클래스 생성시 train data set을 넣어 줘야 하므로 train data set feature을 저장하고 있는 xtrain 리스트와 train data set 답을 저장하고 있는 ytrain 리스트를 받는 인자와 변수가 필요할 것이다. 또한 KNN 알고리즘에서 쓰이는 K값에 해당하는 인자와 변수도 필요하다. 마지막으로 꽃의 이름을 고정시켜 출력할 수 도 있으나 꽃의 이름이 유동적으로 변경되는 것을 고려해 yname 리스트 형식으로 넘기도록 하겠다. 이렇게 총 4개의 생성시 인자를 설정하였다.

- 클래스 변수

dim = xtrain[0].__len__(), disarr =[], vote_cnt = [[0., 0], [0., 1], [0., 2]], grapcor = []

추가적으로 필요한 클래스 변수를 생각해보면 feature 개수(dimension)을 출력하는 show_dim() 함수를 그냥 dimension에 해당하는 변수를 출력하는 것이 보다 편리하므로 dimension을 의미하는 dim 변수를 만들 것이다. 그리고 get_nearest_k() 함수를 구현하기 위한 [거리, 답(꽃 종류)] 를 담고 있는 2차원 리스트 disarr (distance array)와 mv(majority vote)와 wmv(weighted majority vote)에서 계산하는 결과 값 즉, [가중치, 답]을 저장하는 2차원 리스트 vote_cnt(vote count)를 만들 것이다. vote_cnt 리스트 안에 들어있는 리스트는 첫번째 인자가 가중치이며 두번째 인자는 0, 1, 2로 답(꽃의 종류)을 뜻한다. 마지막으로 matplotlib을 이용한 그래프 출력에 필요한 결과값을 K개 만큼 저장했다 출력해야 하므로 mv()와 wmv()가 실행 될 때 마다 grapcor 리스트에 결과를 저장해 줄 것이다.

(2) 클래스 함수 설계

- show_dim(self)

data set에 feature 개수(dimension)를 출력하는 함수이다. 간단히 클래스 변수 dim을 출력하면 될 것이다.

- get_nearest_k(self, xtest_i)

i번째 test data 하나가 들어왔을 때 해당 data를 기준으로 모든 train set data를 봐주면서 거리가 짧은 순으로 리스트 disarr에 저장해주는 함수가 될 것이다. get_nearest_k는 데이터 하나에 대해 계산을 완성해주므로 따로 거리를 구하는 함수를 따로 작성할 필요 없다고 판단 하였다. 구현 방법으로는 일단 모든 train set data를 for문으로 봐주면서 모든 feature 즉, dimension에 대해 (train - test)^2 를 모두 더해주고 루트를 취한 값이 거리가 될 것이다. 이렇게 모든 train data point에 대해 [거리, 답]으로 disarr 리스트에 저장을 해줬다면 최종적으로 [거리, 답]에서 거리를 key 값으로 sorting(오름차순으로)을 해주면 disarr 리스트에는 거리가 짧은 순으로 데이터가 정렬이 된다. 결과적으로 disarr의 첫번째부터 k개의 데이터를 뽑아내면 majority vote를 구현하기 위한 데이터를 뽑아 낼 수 있게 될 것이다.

- mv(self)

majority vote에 해당하는 함수로써 완성된 disarr 리스트 앞부터 k개의 답을 본다. 각각의 가중치는 1로 vote_cnt 리스트에 더해진다. k번 가중치를 더하고 완성된 vote_cnt 리스트를 가지고 가장 큰 가중치를 가진 답을 찾는다. 해당 답이 제일 많이 나온 것이므로 해당 답을 반환하면 될 것이다. 반환할때는 꽃의 이름을 가지고 있는 yname 리스트에 인덱스로 답을 반환하여 꽃의 이름을 반환 하도록 한다.

- wmv(self)

weighted majority vote에 해당하는 함수로써 완성된 disarr 리스트 앞부터 k개의 답을 본다. 각각의 가중치는 1 / distance 로 거리에 반비례하여 가중치를 주도록 하면 될 것이다. 이것 또한 vote_cnt 리스트에 답에 해당하는 가중치를 더해주고 최종적으로 완성된 vote_cnt 리스트에서 가장 큰 가중치를 가진 값을 꽃의 이름으로 반환하면 될 것이다.

- set_k(self, k)

k값을 set 해주는 함수로 k = 3, 5, 10 에 대하여 과제를 시행해야 하므로 이런 함수를 설계하였다.

- get_grapcor(self)

그래프 출력을 하기 위한 리스트 grapcor을 반환해주는 함수가 필요할 것이다.

- reset_grapcor(self)

그래프를 출력하고 난 이후에 grapcor을 reset 해주기 위한 함수가 필요할 것이다.

- reset(self)

다른 데이터가 들어오거나 다음 테스트를 진행해야 할 때 disarr와 vote_cnt 리스트의 초기화가 필요하다. 그것을 해주기 위한 함수이다.

<구현 및 설명>

```python
from operator import itemgetter # 리스트 sorting 시 특정 key 값으로 정렬하기 위함

class KNN():
    def __init__(self, k, xtrain, ytrain, yname):
        self.k = k                                  # KNN 에서의 k 값
        self.xtrain = xtrain                        # train data set feature
        self.ytrain = ytrain                        # train data set 답
        self.yname = yname                          # 꽃이름을 가지고 있는 리스트
        self.dim = xtrain[0].__len__()              # dimension 즉, data set feature 종류
        self.disarr = []                            # distance array 줄임말, [거리, 답] 을 담고있는 리스트
        self.vote_cnt = [[0., 0], [0., 1], [0., 2]] # vote count 줄임말, [가중치, 답] 을 담고있는 리스트 0,1,2 가 얼마나 나왔느냐
        self.grapcor = []                           # 그래프 출력을 하기위해 mv, wmv 의 결과를 순서대로 저장하는 전역 리스트

    def show_dim(self): # 변수 dim 을 출력하는 함수
        print(self.dim)

    def get_nearest_k(self, xtest_i):     # 가장가까운 점부터 순서대로 disarr 리스트에 저장해주는 함수
        for j in range(len(self.xtrain)): # for 문 (train data set 모두에 대하여)
            res = 0.                      # disarr 리스트에 담을 거리 값
            for i in range(0, self.dim):  # dimension 즉, feature 갯수만큼에 대하여 거리를 계산
                res = res + (xtest_i[i] - self.xtrain[j][i])**2 # (train0 - test0)^2 + ... + (train3 - test3)^2
            res = res**0.5                                      # 마지막에 0.5 승으로 루트를 씌워준다.
            self.disarr.append((res, int(self.ytrain[j])))      # disarr에 [res, 답] 을 append 해준다.
        self.disarr.sort(key=itemgetter(0))         # 최종적으로 disarr를 res 기준으로 sorting 한다 (오름차순)

    def mv(self):   # Majority Vote 함수
        for j in range(0, self.k):                          # for 문 (k 값 만큼)
            self.vote_cnt[self.disarr[j][1]][0] += 1        # 가중치는 그냥 갯수이므로 +1 로 해주고, disarr의 0부터 k개를 본다.
        self.vote_cnt.sort(key=itemgetter(0), reverse=True) # vote_cnt에서 가장큰 가중치를 알기 위해 sorting reverse (내림차순)
        self.grapcor.append(self.vote_cnt[0][1])            # 결과를 grapcor 리스트에 넣어준다.
        return self.yname[self.vote_cnt[0][1]]              # 결과를 꽃이름으로 반환한다.
```

```python
    def set_k(self, k):      # k 값을 설정하는 set 함수
        self.k = k

    def get_grapcor(self):   # grapcor 리스트를 반환해주는 get 함수
        return self.grapcor

    def reset_grapcor(self): # grapcor 리스트를 초기화하는 함수
        self.grapcor = []

    def reset(self):         # disarr 리스트와 vote_cnt 리스트를 초기화 하는 함수
        self.disarr = []
        self.vote_cnt = [[0., 0], [0., 1], [0., 2]]
```

클래스명은 KNN으로 설정하였다. 변수는 설계와 같이 8개가 있으며 (k : KNN에서의 k 값, xtrain : train data set feature, ytrain : train data set 답, yname : 꽃 이름, dim : feature 종류 (dimension), disarr : [거리, 답] 을 담고 있는 리스트, vote_cnt : [가중치, 답] 을 담고 있는 리스트, grapcor : 그래프 출력을 위해 순서대로 답을 저장하고 있는 리스트) show_dim(self) 함수는 이 변수중 dim을 출력하는 함수이다. dim의 경우 feature의 개수(종류)를 뜻하므로 train data set feature로 들어온 xtrain 리스트의 하나를 잡고 길이를 재었다. xtrain[0].__len__() 이것을 의미한다.

get_nearst_k(self, xtest_i) 함수는 xtest_i가 테스트 데이터를 의미하며 xtest_i 와 모든 train data

를 비교해주기 위해 train data set인 xtrain 크기만큼 for문을 잡았다. 이후 res = 0. 이라는 변수를 잡고 res에 feature 값의 차이를 제곱한 값을 더해주었다. 코드로는 res = res + (xtest_i[i] - self.xtrain[j][i])**2 이다. 이것을 feature 종류만큼 for문을 돌면서 더해주고 마지막에 res = res**0.5 로 0.5승을 해주어 루트를 씌워주었다. 이로써 거리(distance)를 계산하는 공식을 이용해 거리가 완성이 되었고 이것을 disarr에 [거리, 답] 을 append 해주었다. 모든 train data set에 대하여 거리 계산이 끝나고 disarr가 완성 되었으면 disarr를 거리를 key로 sorting하였다. 코드는 이와 같다. self.disarr.sort(key=itemgetter(0)) 여기서 쓰이는 itemgtter(0)은 다차원 리스트에서 0번째 값을 기준으로 sorting 해준다는 것을 의미하고 이것을 위해 소스코드 상단 맨위에서 from operator import itemgetter을 해주었다.

majority vote에 해당하는 mv(self) 함수는 완성된 disarr 리스트에 앞에서부터 k개의 답만 확인하면 되므로 for문은 k만큼 돌게 구현하였다. for문에서 시행하는 것은 vote_cnt 리스트에 어떤 꽃이 많이 나왔는지 저장해주는 것이며 꽃 종류를 키값으로 찾으며 1씩 더해주도록 구현하였다. 이후 vote_cnt[0]가 가장 높은 가중치를 가지게 하기 위해 vote_cnt 리스트를 내림차순으로 sorting하였다. 여기에 reverse=True 라는 옵션을 사용하였다. sorting을 한 뒤에는 vote_cnt[0][1]가 결과값이므로 이것을 grapcor리스트에 append하고 return 값을 꽃의 이름으로 해주기 위해 yname[vote_cnt[0][1]]을 반환하였다.

weighted majority vote에 해당하는 wmv(self) 함수는 mv(self)와 마찬가지인데 더해주는 가중치를 1로 하는 것이 아니라 (1 / 거리)로 거리가 가까울수록 가중치가 커지게 해주었다. 완성된 disarr 리스트에 앞에서부터 k개의 값을 보기 위해 for문은 k만큼 돌게 하였으며 더해주는 가중치는 self.disarr[j][0]을 통해 disarr에 저장되어있는 거리를 얻어와 1 / self.disarr[j][0] 으로 계산 해주었다. 이 값을 vote_cnt 리스트에 더해주었으며 mv함수와 동일하게 마지막에는 vote_cnt를 내림차순으로 sorting하였다. vote_cnt[0][1]이 결과값이므로 이것을 grapcor리스트에 append하고 return 값은 꽃의 이름으로 해주기 위해 yname[vote_cnt[0][1]]을 반환하였다.

set_k(self, k) 함수는 main에서 k값을 편리하게 설정해주기 위해 만든 set 함수이다. self.k = k

get_grapcor(self) 함수는 mv나 wmv로 완성된 test가 판단하고 있는 답 grapcor 리스트를 반환해주기 위한 get 함수이다. return self.grapcor

reset_grapcor(self) 함수는 그래프 출력이 끝난뒤 초기화가 필요한 grapcor을 비워주는 함수이며 self.grapcor = [ ] 로 구현하였다.

마지막으로 reset(self) 함수는 get_nearest_k 가 끝나고 새로운 test data가 들어오게 될 때 기존 disarr를 초기화 할 필요가 있으므로 disarr를 비워주는 함수이다. vote_cnt도 초기화 해주어야 하므로 vote_cnt는 형식을 유지하고 가중치만 0으로 만들어주는 초기화를 해주었다.

5.2 메인 함수 (main.py) 설계, 구현 및 소스코드

<설계>

과제 설명에 있듯이 iris 데이터를 import하고 그래프 출력을 위해 matplotlib.pyplot 또한 import 해야 할 것이다. iris data set을 불러온 뒤 과제 설명과 같이 iris data set에 15번째 인덱스를 가진 data를 test data set으로 설정하고 나머지를 train data set으로 설정한다.

train data set과 test data set을 리스트에 저장해준 뒤에 k 값을 3, 5, 10 으로 설정해주면서 코드가 실행되게 하기 위해 for문 돌아야 할 것이다. klist = [3, 5, 10] 로 총 세번 돌아주면 될 것이다. 그리고 for문 안에는 test data 10개를 하나씩 넣어주는 for문을 작성하고 최종적으로 majority vote 함수와 weighted majority vote 함수를 순서대로 실행하면 될 것이다. 하지만 중간에 그래프를 출력해야 하므로 majority vote와 weighted majority vote 함수를 실행하는 for문은 나누어 작성해주고 test data set에 대한 for문이 끝날 때 그래프를 출력해주면 될 것이다. 그리고 항상 get_nearest_k 함수를 실행하고 mv나 wmv 함수를 실행하면 reset함수를 써주어 disarr와 vote_cnt 리스트를 초기화 해주어야 할 것이다. 그래프 출력 후에는 grapcor 리스트를 초기화 해주는 reset_grapcor 함수가 사용되어야 한다.

출력결과에 대한 디테일이나 그래프 출력시 label 명시와 title 명시는 코드를 구현하면서 구체화하도록 하겠다.

<구현 및 설명>

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from knnclass import KNN    # knnclass import

iris = load_iris()
X = iris.data[:, :4]        # feature 4종류.
y = iris.target             # 0, 1, 2 로 구성되어있는 데이터
y_name = iris.target_names  # ['setosa' 'versicolor' 'virginica'] 꽃 이름

l = 15
# y.shape[0] = 150 이며 i % l == 14 일때를 test data set 으로 설정한다.
for_test = np.array([(i % l == (l-1)) for i in range(y.shape[0])]) # for_test 는 boolean 값을 담는 리스트가 된다.
for_train = ~for_test       # for_train 은 for_test 의 반대 boolean 값을 담는 리스트가 된다.
# print(len(X_train)) = 140 , print(len(y_train)) = 140
X_train = X[for_train]      # iris.data X에서 for_train 의 true 값만을 담는 리스트가 된다. 크기 140 train data set 의 feature
y_train = y[for_train]      # iris.target y에서 for_train 의 true 값만을 담는 리스트가 된다. 크기 140 train data set 의 답
# print(len(X_test)) = 10, print(len(y_test)) = 10
X_test = X[for_test]        # iris.data X에서 for_test 의 true 값만을 담는 리스트가 된다. 크기 10 test data set 의 feature
y_test = y[for_test]        # iris.target y에서 for_test 의 true 값만을 담는 리스트가 된다. 크기 10 test data set 의 답

print(X_train)

knn_iris = KNN(10, X_train, y_train, y_name)    # KNN class 객체 생성 (k, train set feature, train set 답, 꽃 이름 리스트)
knn_iris.show_dim()             # feature 갯수 출력 (dimension 과 동일 = 4)
klist = [3, 5, 10]              # k = 3, 5, 10 에 대해 돌려주기 위한 리스트
for j in range (0, 3):          # klist 인자로 k = 3, 5, 10 을 돌려주기 위한 for 문
    knn_iris.set_k(klist[j])    # class 객체의 set_k (k 값 설정)

    # Majority Vote
    print("Majority vote / k = ", klist[j])
    for i in range(y_test.shape[0]):        # test data set 에 대한 for 문
        knn_iris.get_nearest_k(X_test[i])   # X_test를 하나씩 넣으며, 가장 가까운 point 10개를 얻고 객체의 리스트에 저장한다.
        # Test Data i 번째 결과를 majority vote 를 통해 출력 , 실제 답(꽃이름)을 y_name[y_test[i]] 으로 출력
        print("Test Data", i, "Computed class:", knn_iris.mv(), ",\tTrue class: ", y_name[y_test[i]])
        knn_iris.reset()                    # 가장 가까운 순으로 정렬된 point 리스트 초기화, majority vote 리스트 초기화

    # test data set 10 개를 모두 실행하면 결과를 저장한 grapcor 리스트를 get_grapcor 을 통해 호출하여 그래프를 출력한다.
    plt.figure(4, figsize=(8, 6))  # matplotlib 출력 화면 크기 지정
    plt.scatter(X_test[:, 0], X_test[:, 1], c=knn_iris.get_grapcor(), cmap=plt.cm.Set1, edgecolor='k')  # 0 수평 1 수직
    plt.title("majority vote / K = " + str(klist[j]))   # 그래프 제목 (알고리즘 / k 값)
    plt.xlabel('Sepal length')  # xlabel 이름
    plt.ylabel('Sepal width')   # ylabel 이름
    plt.xticks(())
    plt.yticks(())
    plt.show()                  # 그래프 출력
    knn_iris.reset_grapcor()    # grapcor 리스트 초기화

    # Weighted Majority Vote
    print("Weigthed Majority vote / k = ", klist[j])
    for i in range(y_test.shape[0]):        # test data set 에 대한 for 문
        knn_iris.get_nearest_k(X_test[i])   # X_test를 하나씩 넣으며, 가장 가까운 point 10개를 얻고 객체의 리스트에 저장한다.
        # Test Data i 번째 결과를 weighted majority vote 를 통해 출력 , 실제 답(꽃이름)을 y_name[y_test[i]] 으로 출력
        print("Test Data", i, "Computed class:", knn_iris.wmv(), ",\tTrue class: ", y_name[y_test[i]])
        knn_iris.reset()                    # 가장 가까운 순으로 정렬된 point 리스트 초기화, majority vote 리스트 초기화

    # test data set 10 개를 모두 실행하면 결과를 저장한 grapcor 리스트를 get_grapcor 을 통해 호출하여 그래프를 출력한다.
    plt.figure(4, figsize=(8, 6))  # matplotlib 출력 화면 크기 지정
    plt.scatter(X_test[:, 0], X_test[:, 1], c=knn_iris.get_grapcor(), cmap=plt.cm.Set1, edgecolor='k')  # 0 수평 1 수직
    plt.title("weighted majority vote / K = " + str(klist[j])) # 그래프 제목 (알고리즘 / k 값)
    plt.xlabel('Sepal length')  # xlabel 이름
    plt.ylabel('Sepal width')   # ylabel 이름
    plt.xticks(())
    plt.yticks(())
    plt.show()                  # 그래프 출력
    knn_iris.reset_grapcor()    # grapcor 리스트 초기화
```

iris data는 iris = load_iris() 를 통해 iris에 저장한뒤 X에는 feature 데이터를, y에는 답에 해당하는 데이터를, y_name에는 꽃의 이름을 담아 주었다. train data set과 test data set을 나누기 위해서 데이터 리스트 안에 인덱스가 15번째 인 것 들을 test data set으로 설정해 주었다. np.array가 쓰이며 i % 15 == 14 인 인덱스로 설정해주었다. 이렇게 되면 for_test 10개의 데이터가 담기는 것이 아니라 Boolean 값을 150개 담고있는 리스트이지만 조건에 맞는 인덱스에 대해서만 True값을 가지는 리스트가 된다. for_train의 경우 for_test와 반대로 설정해주면 되므로 for_train = ~for_test 로 이것을 설정해 주었다. 이렇게 완성된 Boolean 리스트를 이용해 실제 train data set과 test data set을 X_train = X[for_train] , X_test = X[for_test] 를 통해 설정해 주었다. 실제로 X_train의 길이를 len을 이용해 print 해볼 경우 140이 나오고 X_test의 길이는 10이 나오는 것을 관찰할 수 있었다.

data set 설정 이후에는 knn_iris라는 이름으로 KNN 클래스 객체를 생성하고 train data set feature와 답을 생성자 인자로 넣어준다. show_dim() 함수를 호출해서 feature 종류를 출력해서 확인하고 klist = [3, 5, 10] 리스트를 for문을 통해 돌면서 k 값을 바꾸면서 프로그램을 실행할 수 있게 한다. k 값은 KNN 클래스의 set_k 함수를 통해 설정해주었고 이후 majority vote, 그래프 출력, weighted majority vote, 그래프 출력 순으로 코드를 작성하였다. mv() 함수는 test data set 크기만큼 for문을 돌며 get_nearest_k() 함수를 실행하고 mv() 를 호출하여 계산하였다 mv()나 wmv() 함수의 경우 return 값이 꽃 이름이므로 print문 안에 호출하여 바로 출력될 수 있게끔 하였다. 이렇게 test data set 에 대한 for문이 끝나면 추정한 답을 순서대로 저장하고있는 grapcor 리스트를 get_grapcor() 함수로 호출하여 그래프를 출력하였다. 그래프의 제목과 xlabel, ylabel을 설정해 주었으며 plt.show()를 통해 그래프 출력이 끝난 이후에는 이후 그래프 출력을 위해서 grapcor 리스트를 reset_grapcor() 함수로 초기화 해주었다.

6. 학습 과정에 대한 설명

학습 과정이 제대로 진행되고 있는지 보기 위하여 get_nearest_k() 함수가 실행 된 후 가까운 거리 순으로 disarr 리스트가 정렬되었는지 print 해보고 k개의 개수를 제대로 보고 majority vote와 weighted majority vote 결과를 뽑아내는지 vote_cnt 리스트와 답을 출력해 보았다.

```
Majority vote / k =  3
거리,답 disarr리스트 /  [(0.412310562561766, 0), (0.46904157598234253, 0), (0.5477225575051664, 0), (0.5567764362830022, 0),
vote결과 vote_cnt리스트 [[3.0, 0], [0.0, 1], [0.0, 2]]
책정 답 :  setosa
Test Data 0 Computed class: setosa ,    True class:  setosa
거리,답 disarr리스트 /  [(0.1414213562373093, 0), (0.17320508075688812, 0), (0.22360679774997858, 0), (0.22360679774997935,
vote결과 vote_cnt리스트 [[3.0, 0], [0.0, 1], [0.0, 2]]
책정 답 :  setosa
Test Data 1 Computed class: setosa ,    True class:  setosa
거리,답 disarr리스트 /  [(0.3605551275463988, 0), (0.3741657386773947, 0), (0.41231056256176585, 0), (0.41231056256176596, 0
vote결과 vote_cnt리스트 [[3.0, 0], [0.0, 1], [0.0, 2]]
책정 답 :  setosa
Test Data 2 Computed class: setosa ,    True class:  setosa
거리,답 disarr리스트 /  [(0.5099019513592782, 1), (0.5196152422706634, 1), (0.529150262212918, 1), (0.5385164807134499, 1),
vote결과 vote_cnt리스트 [[3.0, 1], [0.0, 0], [0.0, 2]]
책정 답 :  versicolor
```

```
Weigthed Majority vote / k =  3
거리,답 disarr리스트 /  [(0.412310562561766, 0), (0.46904157598234253, 0), (0.5477225575051664, 0), (0.556776436
vote결과 vote_cnt리스트 [[6.383105272269988, 0], [0.0, 1], [0.0, 2]]
책정 답 :  setosa
Test Data 0 Computed class: setosa ,    True class:  setosa
거리,답 disarr리스트 /  [(0.1414213562373093, 0), (0.17320508075688812, 0), (0.22360679774997858, 0), (0.223606
vote결과 vote_cnt리스트 [[17.316706458761317, 0], [0.0, 1], [0.0, 2]]
책정 답 :  setosa
Test Data 1 Computed class: setosa ,    True class:  setosa
거리,답 disarr리스트 /  [(0.3605551275463988, 0), (0.3741657386773947, 0), (0.41231056256176585, 0), (0.412310
vote결과 vote_cnt리스트 [[7.871469650613717, 0], [0.0, 1], [0.0, 2]]
책정 답 :  setosa
Test Data 2 Computed class: setosa ,    True class:  setosa
거리,답 disarr리스트 /  [(0.5099019513592782, 1), (0.5196152422706634, 1), (0.529150262212918, 1), (0.538516480
vote결과 vote_cnt리스트 [[5.7754846137267295, 1], [0.0, 0], [0.0, 2]]
책정 답 :  versicolor
Test Data 3 Computed class: versicolor ,    True class:  versicolor
```

```
Weigthed Majority vote / k =  5
거리,답 disarr리스트 /  [(0.412310562561766, 0), (0.46904157598234253, 0), (0.5477225575051664, 0), (0.5567764362830022, 0), (0.5830951894845297, 0), (0.59160
vote결과 vote_cnt리스트 [[9.894144143962826, 0], [0.0, 1], [0.0, 2]]
책정 답 :  setosa
Test Data 0 Computed class: setosa ,    True class:  setosa
거리,답 disarr리스트 /  [(0.1414213562373093, 0), (0.17320508075688812, 0), (0.22360679774997858, 0), (0.22360679774997935, 0), (0.24494897427831802, 0), (0.2
vote결과 vote_cnt리스트 [[25.871325318399517, 0], [0.0, 1], [0.0, 2]]
책정 답 :  setosa
Test Data 1 Computed class: setosa ,    True class:  setosa
거리,답 disarr리스트 /  [(0.3605551275463988, 0), (0.3741657386773947, 0), (0.41231056256176585, 0), (0.41231056256176596, 0), (0.47958315233127163, 0), (0.499
vote결과 vote_cnt리스트 [[12.381970041547797, 0], [0.0, 1], [0.0, 2]]
책정 답 :  setosa
Test Data 2 Computed class: setosa ,    True class:  setosa
거리,답 disarr리스트 /  [(0.5099019513592782, 1), (0.5196152422706634, 1), (0.529150262212918, 1), (0.5385164807134499, 1), (0.5477225575051655, 1), (0.547722
vote결과 vote_cnt리스트 [[9.458179853847806, 1], [0.0, 0], [0.0, 2]]
책정 답 :  versicolor
Test Data 3 Computed class: versicolor ,    True class:  versicolor
거리,답 disarr리스트 /  [(0.20000000000000018, 1), (0.2645751311064587, 1), (0.3605551275463984, 1), (0.3872983346207415, 1), (0.38729833462074176, 1), (0.412
vote결과 vote_cnt리스트 [[16.717123506161645, 1], [0.0, 0], [0.0, 2]]
```

disarr 리스트에는 성공적으로 거리가 정렬되었고, vote_cnt 리스트에 계산이 올바르게 들어가 그것을 보고 결과를 책정했음을 알 수 있다. k=3 이외 k=5, k=10에 대해서도 올바르게 정렬되고 계산된 가중치가 들어가고 있음을 확인하였다.

7. 결과 및 분석

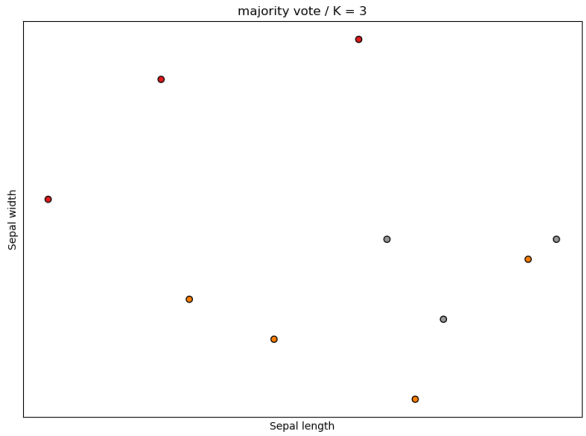
7.1 결과

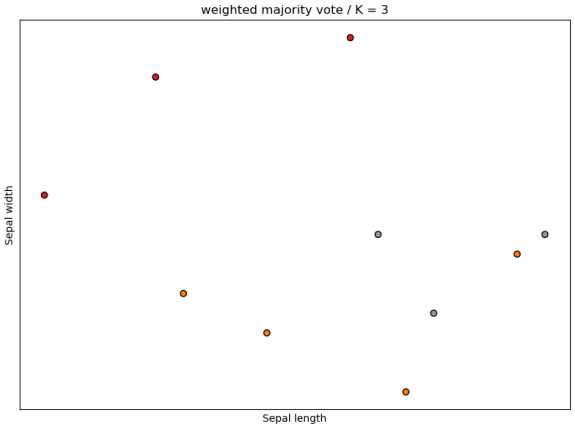
```
4
Majority vote / k =  3
Test Data 0 Computed class: setosa ,    True class:  setosa
Test Data 1 Computed class: setosa ,    True class:  setosa
Test Data 2 Computed class: setosa ,    True class:  setosa
Test Data 3 Computed class: versicolor ,    True class:  versicolor
Test Data 4 Computed class: versicolor ,    True class:  versicolor
Test Data 5 Computed class: versicolor ,    True class:  versicolor
Test Data 6 Computed class: virginica , True class:  virginica
Test Data 7 Computed class: versicolor ,    True class:  virginica
Test Data 8 Computed class: virginica , True class:  virginica
Test Data 9 Computed class: virginica , True class:  virginica
```

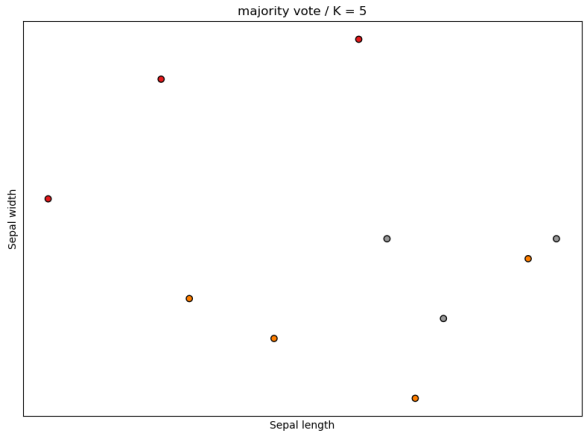
```
Weigthed Majority vote / k =  3
Test Data 0 Computed class: setosa ,    True class:  setosa
Test Data 1 Computed class: setosa ,    True class:  setosa
Test Data 2 Computed class: setosa ,    True class:  setosa
Test Data 3 Computed class: versicolor ,    True class:  versicolor
Test Data 4 Computed class: versicolor ,    True class:  versicolor
Test Data 5 Computed class: versicolor ,    True class:  versicolor
Test Data 6 Computed class: virginica , True class:  virginica
Test Data 7 Computed class: versicolor ,    True class:  virginica
Test Data 8 Computed class: virginica , True class:  virginica
Test Data 9 Computed class: virginica , True class:  virginica
Majority vote / k =  5
Test Data 0 Computed class: setosa ,    True class:  setosa
Test Data 1 Computed class: setosa ,    True class:  setosa
Test Data 2 Computed class: setosa ,    True class:  setosa
Test Data 3 Computed class: versicolor ,    True class:  versicolor
Test Data 4 Computed class: versicolor ,    True class:  versicolor
Test Data 5 Computed class: versicolor ,    True class:  versicolor
Test Data 6 Computed class: virginica , True class:  virginica
Test Data 7 Computed class: versicolor ,    True class:  virginica
Test Data 8 Computed class: virginica , True class:  virginica
Test Data 9 Computed class: virginica , True class:  virginica
```

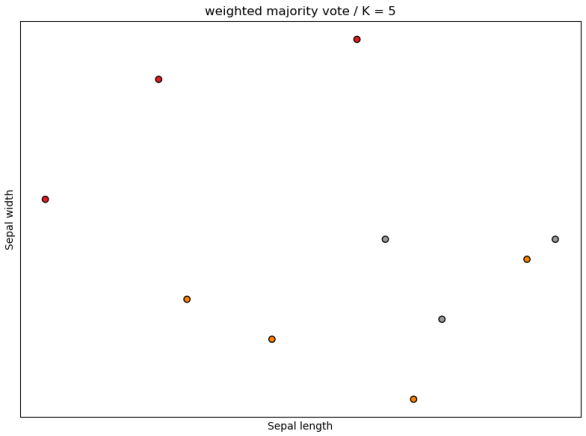
```
Weigthed Majority vote / k =  5
Test Data 0 Computed class: setosa ,    True class:  setosa
Test Data 1 Computed class: setosa ,    True class:  setosa
Test Data 2 Computed class: setosa ,    True class:  setosa
Test Data 3 Computed class: versicolor ,    True class:  versicolor
Test Data 4 Computed class: versicolor ,    True class:  versicolor
Test Data 5 Computed class: versicolor ,    True class:  versicolor
Test Data 6 Computed class: virginica , True class:  virginica
Test Data 7 Computed class: versicolor ,    True class:  virginica
Test Data 8 Computed class: virginica , True class:  virginica
Test Data 9 Computed class: virginica , True class:  virginica
Majority vote / k =  10
Test Data 0 Computed class: setosa ,    True class:  setosa
Test Data 1 Computed class: setosa ,    True class:  setosa
Test Data 2 Computed class: setosa ,    True class:  setosa
Test Data 3 Computed class: versicolor ,    True class:  versicolor
Test Data 4 Computed class: versicolor ,    True class:  versicolor
Test Data 5 Computed class: versicolor ,    True class:  versicolor
Test Data 6 Computed class: virginica , True class:  virginica
Test Data 7 Computed class: virginica , True class:  virginica
Test Data 8 Computed class: virginica , True class:  virginica
Test Data 9 Computed class: virginica , True class:  virginica
```

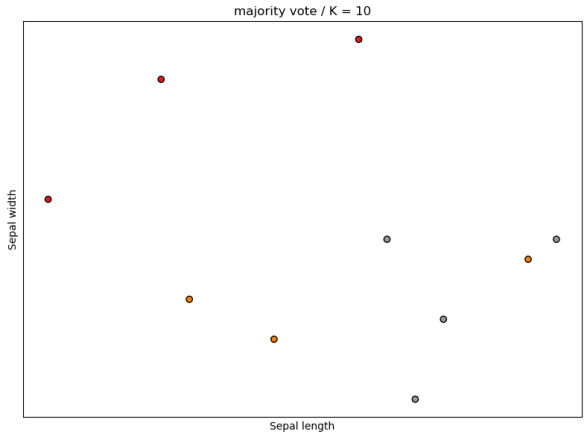
```
Weigthed Majority vote / k =  10
Test Data 0 Computed class: setosa ,    True class:  setosa
Test Data 1 Computed class: setosa ,    True class:  setosa
Test Data 2 Computed class: setosa ,    True class:  setosa
Test Data 3 Computed class: versicolor ,    True class:  versicolor
Test Data 4 Computed class: versicolor ,    True class:  versicolor
Test Data 5 Computed class: versicolor ,    True class:  versicolor
Test Data 6 Computed class: virginica , True class:  virginica
Test Data 7 Computed class: virginica , True class:  virginica
Test Data 8 Computed class: virginica , True class:  virginica
Test Data 9 Computed class: virginica , True class:  virginica
```

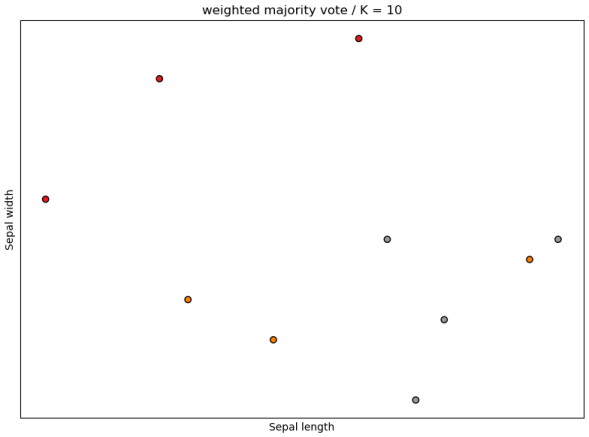










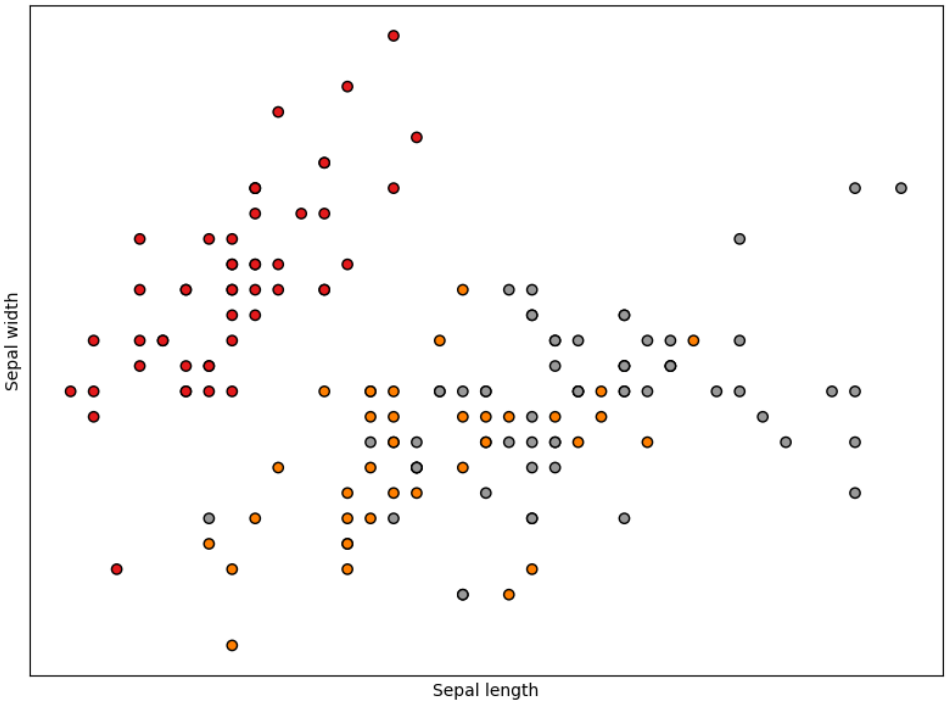


k = 3, 5, 10 에 대해 majority vote 와 weighted majority vote 가 모두 동일한 결과를 보여주었다. k = 3, 5 는 출력 결과를 보다시피 Test Data 7번째에 대해 versicolor를 계산해내지만 실제 답은 virginica로 7번째 데이터는 틀린다. k = 10 은 모든 데이터에 대해 정답을 출력해 내는 것을 볼 수 있다.

7.2 분석

사실 majority vote와 weighted majority vote 차이를 관찰하고 싶었지만 mv함수와 wmv함수의 결과가 동일하게 나오는 아쉬움이 있었다. 그렇다면 이 data set의 경우 mv와 wmv간의 차이가 거의 없는 data set 일수 있다는 추측이 가능하고 그러면 데이터가 그만큼 직관적이게 구성되어 있다고 추론할 수 있다. 그 이유로는 mv와 wmv결과가 다르게 나올려면 예외적인 상황이 필요하다. 예를 들어 majority vote에서 작은 A그룹이 B그룹에 둘러 쌓여 있을 경우 K값이 크게 되면 A그룹 안에 test data가 주어지더라도 test 는 B그룹으로 판단될 것이다.

그렇다면 실제 train data set이 어떻게 구성되어 있는지 보도록 하자



위 그림과 같이 데이터가 어느정도 직관적으로 주어져 있음을 관찰 할 수 있다. 물론 주황색과 회색이 섞여 있는 구간에 데이터가 주어진다면 결과가 상이 할 수 있다. 하지만 프로그램 출력 결과에서 test data set을 보면 주황색과 회색이 섞여 있는 구간에 있는 test data가 없음을 알 수 있다.

결과적으로 data의 답이 정해져 있는 경우 알고리즘이 좋더라도 답이 틀리면 알고리즘을 수정해야 하는 것이다. 즉, weighted majority vote의 답은 정해져 있는 것이 아니라 data set에 따라서 wmv에서 거리에 따라 주는 가중치 공식을 적절히 할 필요가 있는 것이다.