기계학습 4주차 과제 보고서

12191656 이채연

1. 기준점 좌표 값 3개 설정

먼저 기준점 좌표 값 x1, x2, x3 3개를 설정하였다. 이때 rand함수를 사용해서 임의의 값으로 설정되도록 하였다.

In [2]:
```
K = 10
rand_data = np.random.rand(2,100)
x1 = np.random.rand(2,1)
x2 = np.random.rand(2,1)
x3 = np.random.rand(2,1)
```

2. 유클리디언 거리(Euclidean distance)를 거리 함수로 KNN 알고리즘에 적용

데이터와 기준점 간 유클리디언 거리를 계산하는 공식을 numpy로 구현한 파이썬 코드는 다음과 같다.

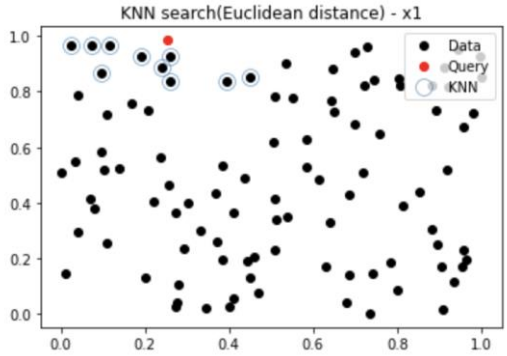
기준점이 3개이므로 distance1, distance2, distance3에 각각의 기준점과 데이터 간 거리를 계산한 결과값을 넣어주었다.

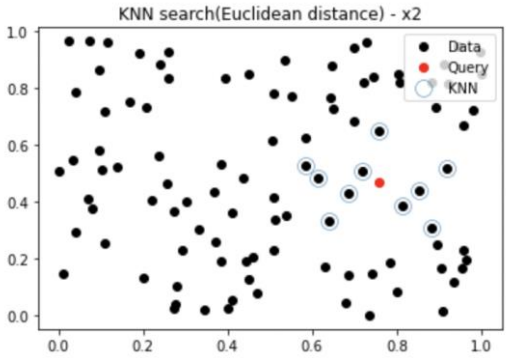
In [4]:
```
distance1 = np.sqrt(((rand_data-x1)**2).sum(axis=0))
distance2 = np.sqrt(((rand_data-x2)**2).sum(axis=0))
distance3 = np.sqrt(((rand_data-x3)**2).sum(axis=0))
# Sort the distances
idx1 = np.argsort(distance1)
idx2 = np.argsort(distance2)
idx3 = np.argsort(distance3)
# return the indices of K nearest neighbor
neighborIndex1 = idx1[:K]
neighborIndex2 = idx2[:K]
neighborIndex3 = idx3[:K]
distance1
```

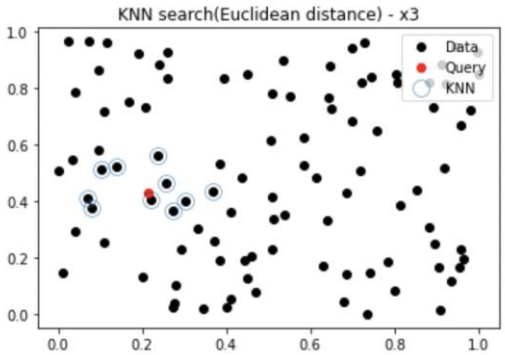
다음은 KNN 결과를 시각화하는 코드이다. 기준점 x1, x2, x3에 대해, 그리고 유클리디언 거리, 맨해튼 거리, 코사인 거리에 대해 모두 동일하게 수행하였다.

In [5]:
```
plt.plot(rand_data[0, :], rand_data[1, :], "ok", label="Data")
plt.plot(x1[0,0], x1[1,0], 'or', label='Query')
plt.plot(rand_data[0, neighborIndex1], rand_data[1, neighborIndex1], 'o',
        markerfacecolor='None', markersize=12,
        markeredgewidth = 0.5, label = 'KNN')
plt.legend(loc='upper right')
plt.title('KNN search(Euclidean distance) - x1')
plt.show()
```

- **KNN 결과 시각화 수행**





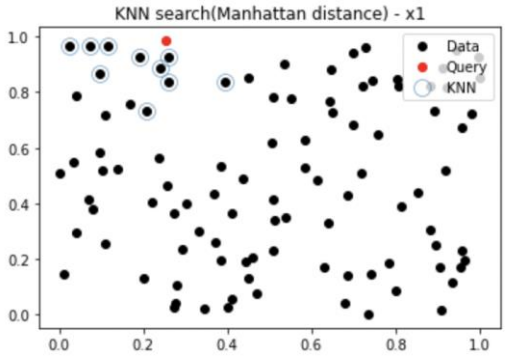


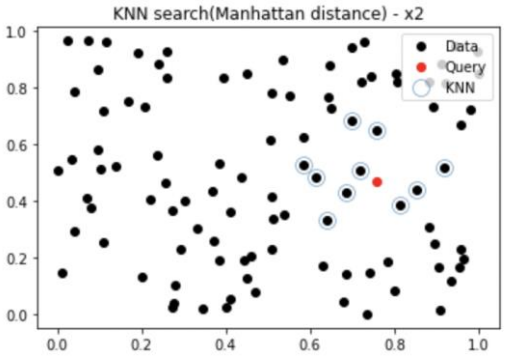
3. 맨해튼 거리(Manhattan distance)를 거리 함수로 KNN 알고리즘에 적용

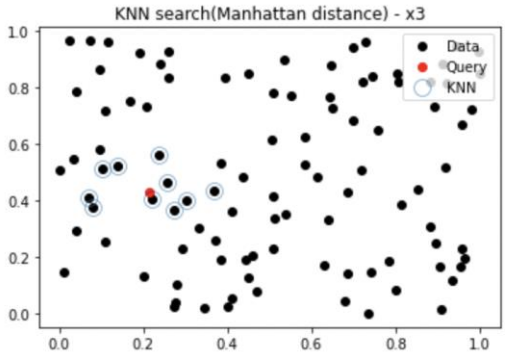
데이터와 기준점 간 맨해튼 거리를 계산하는 공식을 numpy로 구현한 파이썬 코드는 다음과 같다. 기준점이 3개이므로 distance1, distance2, distance3에 각각의 기준점과 데이터 간 거리를 계산한 결과값을 넣어주었다.

In [8]:
```
distance1 = abs(rand_data-x1).sum(axis=0)
distance2 = abs(rand_data-x2).sum(axis=0)
distance3 = abs(rand_data-x3).sum(axis=0)
# Sort the distances
idx1 = np.argsort(distance1)
idx2 = np.argsort(distance2)
idx3 = np.argsort(distance3)
# return the indices of K nearest neighbor
neighborIndex1 = idx1[:K]
neighborIndex2 = idx2[:K]
neighborIndex3 = idx3[:K]
```

- **KNN 결과 시각화 수행**





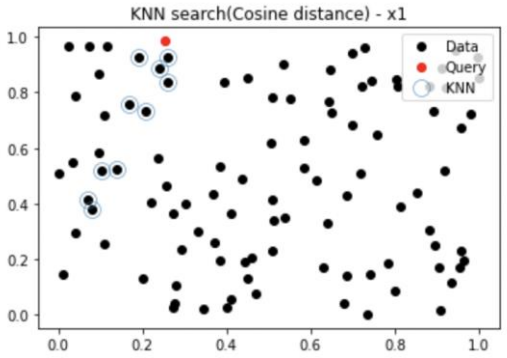


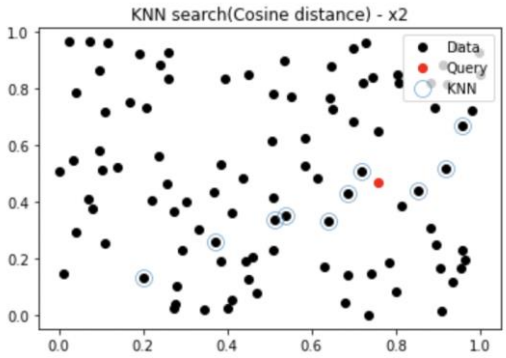
4. 코사인 거리(Cosine distance)를 거리 함수로 KNN 알고리즘에 적용

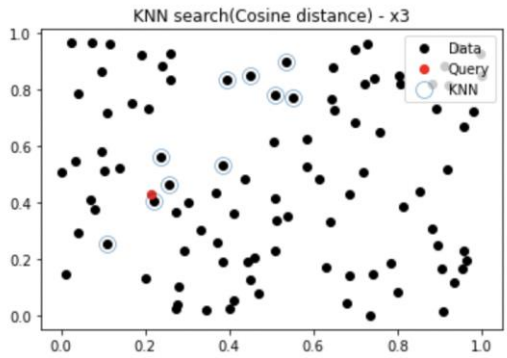
데이터와 기준점 간 코사인 거리를 계산하는 공식을 numpy로 구현한 파이썬 코드는 다음과 같다. x1, x2, x3의 shape이 2*1이고 rand_data의 shape이 2*100이므로 이 둘을 dot product하기 위해서 x1,x2,x3를 transpose한 뒤 1*2 로 reshape해주었다. 기준점이 3개이므로 distance1, distance2, distance3에 각각의 기준점과 데이터 간 거리를 계산한 결과값을 넣어주었다.

In [12]:
```
l2_x1 = np.sqrt(sum(np.square(x1)))
l2_rand_data = np.sqrt(sum(np.square(rand_data)))
distance1 = 1 - np.dot(x1.T.reshape(2,), rand_data)/(l2_x1 * l2_rand_data)

l2_x2 = np.sqrt(sum(np.square(x2)))
distance2 = 1 - np.dot(x2.T.reshape(2,), rand_data)/(l2_x2 * l2_rand_data)

l2_x3 = np.sqrt(sum(np.square(x3)))
distance3 = 1 - np.dot(x3.T.reshape(2,), rand_data)/(l2_x3 * l2_rand_data)

# Sort the distances
idx1 = np.argsort(distance1)
idx2 = np.argsort(distance2)
idx3 = np.argsort(distance3)
# return the indices of K nearest neighbor
neighborIndex1 = idx1[:K]
neighborIndex2 = idx2[:K]
neighborIndex3 = idx3[:K]
```

- **KNN 결과 시각화 수행**


KNN search(Cosine distance) - x1
Data
Query
KNN
0.0
0.2
0.4
0.6
0.8
1.0


KNN search(Cosine distance) - x2
Data
Query
KNN
0.0
0.2
0.4
0.6
0.8
1.0


KNN search(Cosine distance) - x3
Data
Query
KNN
0.0
0.2
0.4
0.6
0.8
1.0