지능정보수학 기말고사대체과제 Part 1

휴먼지능정보공학과 201810825 홍종현

1. 접근

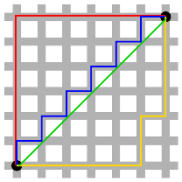
Nearest Neighbor 알고리즘을 구현하는 문제이기 때문에, 문제해결 이전에 Nearest Neighbor에 대해 알아야한다. Nearest Neighbor이란 최근접 이웃법 이라고도 하며 새로운 Data를 입력받았을 때 그 데이터가 좌표상(N차원의)에서 가장 가까이 있는 것이 무엇이냐를 중심으로 새로운 Data의 종류를 정해주는 알고리즘이다. 하지만 모든 DataSet이 정확하게 classification 되어있어 Data들의 위치집합이 정확한 특성을 가지고 있지않다. 때문에 새로운 Data가 기존 DataSet과 비교하였을 때 가장 가까이 있는것과 같게 classification하는 것은 정확도가 매우 떨어진다. 때문에 주변에 무엇이 가장 가까이 있는것인가? 를 보는게 아닌 주변에 있는 몇 개의 것들을 봐서 가장 많은 것을 골라내는 방식을 사용한다. 이 방식을 K- Nearest Neighbor이라고 하며 이때의 K는 주변에서 확인할 Data들의 개수를 뜻한다.

대체과제 공지를 보면 Eucildean distance, Manhattan distance 버전으로 구현하게끔 되어있기때문에 각각이 무엇을 뜻하는지, 어떤 방식으로 구현해야하는지 또한 알아야 한다.
먼저 Euclidean distance란 두점사이의 거리를 구하는것과 같은 말이다. 물론 주어진 Data들의 feature가 2개일 경우 같고, 그 이상 N개의 경우에는 각 차원의 차이만 더해주면 되는 간단한 방식이다.

$$\sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\ \dots\ +(q_n-p_n)^2}=\sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}$$

위 식은 N개의 feature(N개의 차원)가 존재할때의 Euclidean distance를 구하는 방법이다

Manhattan distance란 다음 그림 하나로 설명이 가능하다.



좌하단 점을 시작점, 우상단 점을 도착점이라 하였을 때, 녹색선이 Eucildean distance, 나머지 빨간색, 파란색, 노란색 은 모두 Manhattan distance이다. Manhattan distance를 구하는 식은 |X1-X2| + |Y1-Y2|로 구할 수 있다. 만약 2개의 feature가 아닌 N개의 feature라면(N개의 차원) 그만큼 절댓값들의 차를 더해주면 된다.

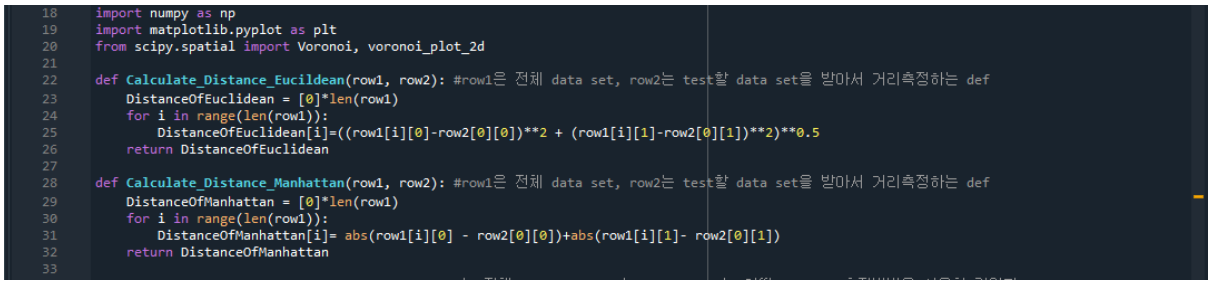
$$d(A,B)=\sum_{i=1}^{n}|a_i-b_i|$$

위 식은 N개의 feature(N개의 차원)가 존재할 때의 Manhattan distance를 구하는 방법이다.

2. 구현

Nearest Neighbor 구현 코드는 크게 6가지 부분으로 나눌 수 있다. Eucildean, Manhattan Distance를 구하는 def, K- Nearest Neighbor로 distance가 작은 순으로 정렬된 K개의 data를 구하는 def, 그 data의 실제 결과를 보여주는 def, Voronoi tessellation을 그리는 def, 실제 구현부로 이루어져있다.

코드를 가져와 주석처리 하면서 설명하기에는 길기 때문에 각 메소드마다 구분하여 설명하겠다.

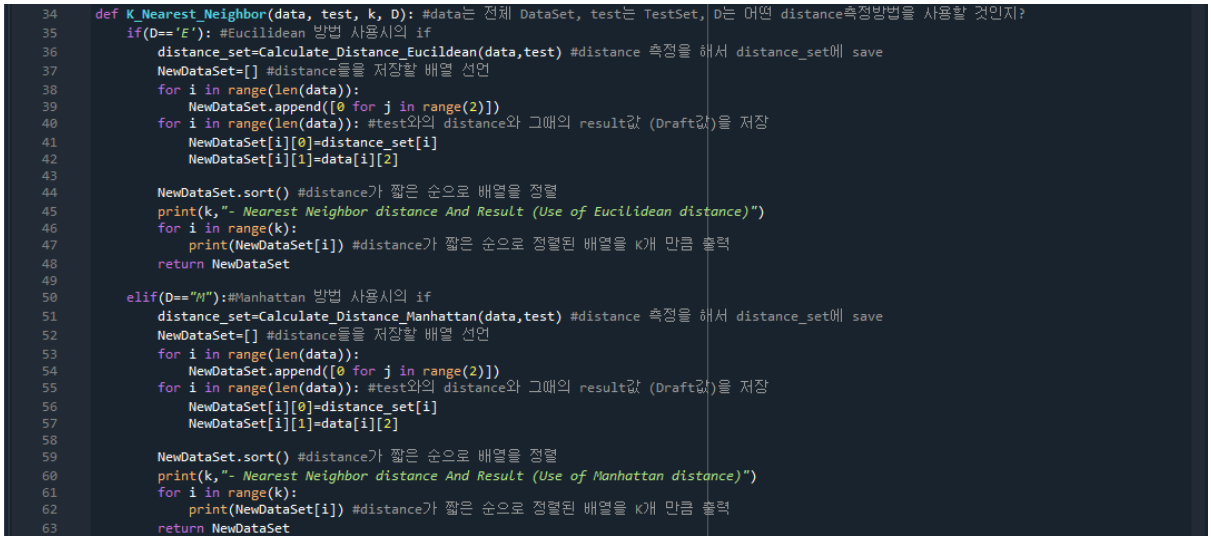


```
18  import numpy as np
19  import matplotlib.pyplot as plt
20  from scipy.spatial import Voronoi, voronoi_plot_2d
21
22  def Calculate_Distance_Eucildean(row1, row2): #row1은 전체 data set, row2는 test할 data set을 받아서 거리측정하는 def
23      DistanceOfEuclidean = [0]*len(row1)
24      for i in range(len(row1)):
25          DistanceOfEuclidean[i]=((row1[i][0]-row2[0][0])**2 + (row1[i][1]-row2[0][1])**2)**0.5
26      return DistanceOfEuclidean
27
28  def Calculate_Distance_Manhattan(row1, row2): #row1은 전체 data set, row2는 test할 data set을 받아서 거리측정하는 def
29      DistanceOfManhattan = [0]*len(row1)
30      for i in range(len(row1)):
31          DistanceOfManhattan[i]= abs(row1[i][0] - row2[0][0])+abs(row1[i][1]- row2[0][1])
32      return DistanceOfManhattan
33
```

line18~lin20은 코드 구현시에 필연적인 lib들을 추가하는 코드

line22은 Eucildean distance를 구하는 메소드이다. row1, row2 두좌표를 입력받아 DistanceOf Eucildean 배열에 순서대로 값을 저장한다. 단순히 거리만 계산하고 값 저장은 외부에서도 할 수 있었지만 list 관련 오류가 발생하여 부득이하게 메소드 안에서 해결하였다.

line28는 Manhattan distance를 구하는 메소드이다. Eucildean distance와 같은방식으로 작동한다. 메소드안의 계산방식만 변화하였다.



```
34  def K_Nearest_Neighbor(data, test, k, D): #data는 전체 DataSet, test는 TestSet, D는 어떤 distance측정방법을 사용할 것인지?
35      if(D=='E'): #Eucildean 방법 사용시의 if
36          distance_set=Calculate_Distance_Eucildean(data,test) #distance 측정을 해서 distance_set에 save
37          NewDataSet=[] #distance들을 저장할 배열 선언
38          for i in range(len(data)):
39              NewDataSet.append([0 for j in range(2)])
40          for i in range(len(data)): #test와의 distance와 그때의 result값 (Draft값)을 저장
41              NewDataSet[i][0]=distance_set[i]
42              NewDataSet[i][1]=data[i][2]
43
44          NewDataSet.sort() #distance가 짧은 순으로 배열을 정렬
45          print(k,"- Nearest Neighbor distance And Result (Use of Eucilidean distance)")
46          for i in range(k):
47              print(NewDataSet[i]) #distance가 짧은 순으로 정렬된 배열을 K개 만큼 출력
48          return NewDataSet
49
50      elif(D=="M"):#Manhattan 방법 사용시의 if
51          distance_set=Calculate_Distance_Manhattan(data,test) #distance 측정을 해서 distance_set에 save
52          NewDataSet=[] #distance들을 저장할 배열 선언
53          for i in range(len(data)):
54              NewDataSet.append([0 for j in range(2)])
55          for i in range(len(data)): #test와의 distance와 그때의 result값 (Draft값)을 저장
56              NewDataSet[i][0]=distance_set[i]
57              NewDataSet[i][1]=data[i][2]
58
59          NewDataSet.sort() #distance가 짧은 순으로 배열을 정렬
60          print(k,"- Nearest Neighbor distance And Result (Use of Manhattan distance)")
61          for i in range(k):
62              print(NewDataSet[i]) #distance가 짧은 순으로 정렬된 배열을 K개 만큼 출력
63          return NewDataSet
```

line 34는 K-Nearest Neighbor 방식을 실제로 적용하는 메소드이다. 파라미터의 data는 전체 DataSet을, test는 주어진 DataSet과 거리를 측정할 하나의 TestSet, K는 K-Nearest Neighbor에서 사용할 K, D는 distance의 측정 방법을 정하는 문자열이다.
line35와 line50에서 distance의 측정방법이 Euclidean distance인지(E로 판별), Manhattan distance 인지(M으로 판별) 분류하여 NewDataSet이라는 배열에 기존 DataSet과 TestSet의 거리, 그리고 그 때의 result값 (여기서는 Draft값)을 저장한다. (이때 Draft가 Yes이면 1로, No이면 0으로 본다)
line44, line59에서 새롭게 생성되어 값이 들어간 NewDataSet을 distance가 짧은 순으로 정렬한다.
line45~48, line60~63에서 distance가 짧은 순으로 정렬된 배열을 K개만큼 출력하고 NewDataSet을 return 한다.

```
64
65  def Show_Result(distance_set, k): #실제 결과를 보여주는 def
66      count=0 #classification 결과가 1일때 저장할 변수
67      result=2 #최종 결과 (Draft)를 0또는 1로 저장하기 위해 2로 초기화
68      for i in range(k):
69          if(distance_set[i][1]==1):
70              count+=1
71      if(count>int(k/2)): #전체 count 수 중에서 과반수 이상일 경우(K개로 추출했을때 K의 과반수)
72          #print("This requirement's classification is Yes(True)\n")
73          result=1 #결과값 1저장
74          return result
75      else:
76          #print("This requirement's classification is No(Flase)\n")
77          result=0 #결과값 0저장
78          return result
79
```

line65은 실제 결과를 보여주는 메소드이다. line66에서 classification의 결과가 1일 때마다 count할 변수를 생성하고, 실제 결과를 저장하기위한 result 변수를 0,1을 저장하기 위해 2로 초기화한다.
line68에서 실제 classification되어 K개만큼 잘랐을 때 그때의 result값이 (Draft 결과) 1인경우를 count해서 누적덧셈한다.
line71~74, line75~78은 총 count수 중에서 K의 과반수를 넘으면 True(Yes)로, 안넘으면 False(No)로 result에 저장하여 result 반환

```
80  def Print_Voronoi(DataSet,K,distance):
81      D=np.array(DataSet)
82      x1=np.array(D[:,0])
83      x2=np.array(D[:,1])
84      y=np.array(D[:,2])
85      points=np.delete(DataSet,2,axis=1)
86      vor = Voronoi(points)
87      voronoi_plot_2d(vor, show_vertices=False, line_width=2, line_alpha=0.6, point_size=2)
88      if(K==3 and distance =="E"):
89          qtitle = "3NN with Euclidean"
90      elif(K==3 and distance == "M"):
91          qtitle = "3NN with Manhattan"
92      elif(K==5 and distance == "E"):
93          qtitle = "5NN with Euclidean"
94      elif(K==5 and distance == "M"):
95          qtitle = "5NN with Manhattan"
96      plt.title(qtitle)
97      for i in range(len(y)):
98          if y[i] == 1:
99              plt.plot(x1[i],x2[i],'ob')
100         elif y[i] == 0:
101             plt.plot(x1[i],x2[i],'xr')
102     plt.show()
103
```

line80은 Voronoi tessellation을 그리는 메소드다.
line81에서 DataSet을 array형식으로 변환 후 Speed를 x1에, Agility를 x2에, 결과값을 y에 각각 나눠 담았다.
그후 Voronoi tessellation을 작성 후에 line88에서 classification 된 결과에 따라 Yes이면 파란색 원으로, No이면 빨간색 X로 표시하게 하였다.

```
104  def Show_KNN(k,D):
105      DataSet=[  #초기 DataSet
106      [2.50, 6.00, 0],[3.75, 8.00, 0],[2.25, 5.50, 0],[3.25, 8.25, 0],
107      [2.75, 7.50, 0],[4.50, 5.00, 0],[3.50, 5.25, 0],[3.00, 3.25, 0],
108      [4.00, 4.00, 0],[4.25, 3.75, 0],[2.00, 2.00, 0],[5.00, 2.50, 0],
109      [8.25, 8.50, 0],[5.75, 8.75, 1],[4.75, 6.25, 1],[5.50, 6.75, 1],
110      [5.25, 9.50, 1],[7.00, 4.25, 1],[7.50, 8.0, 1],[7.25, 5.75, 1]
111      ]
112
113      TestSet=[[6.75, 3],[5.34, 6.0],[4.67,8.4],[7.0,7.0],[7.8,5.4]] #Test할 TestSet들 (classification 후에 DataSet에 추가될 예정)
114      for i in range(5):
115          Q=K_Nearest_Neighbor(DataSet, [TestSet[i]],k,D)
116          if(Show_Result(Q, k)==1):
117              if(i==0): #i번째 순서에 맞게 DataSet에 추가
118                  DataSet.append([6.75, 3,1])
119              elif(i==1):
120                  DataSet.append([5.34, 6.0,1])
121              elif(i==2):
122                  DataSet.append([4.67, 8.4,1])
123              elif(i==3):
124                  DataSet.append([7.0, 7.0,1])
125              elif(i==4):
126                  DataSet.append([7.8, 5.4,1])
127              print(TestSet[i],"'s classification is Yes(True) \n")
128          elif(Show_Result(Q, k)==0):
129              if(i==0): #i번째 순서에 맞게 DataSet에 추가
130                  DataSet.append([6.75, 3,0])
131              elif(i==1):
132                  DataSet.append([5.34, 6.0,0])
133              elif(i==2):
134                  DataSet.append([4.67, 8.4,0])
135              elif(i==3):
136                  DataSet.append([7.0, 7.0,0])
137              elif(i==4):
138                  DataSet.append([7.8, 5.4,0])
139              print(TestSet[i],"'s classification is No(False) \n")
140      #print(k,"- NN(with",D,") classfications's DataSet is ",DataSet)
141      Print_Voronoi(DataSet,k,D)
142
```

line104는 실제 KNN을 실행하였을 때 결과를 보여주는 메소드다

파라미터로 KNN에서 K값과 그때의 Distance 측정방법(Eucildean, Manhattan)을 받아서

line115을 통해 주어진 파라미터의 값으로 KNN을 실행한다.

line116~126,128~138에서 classification 되어 return되는 1또는 0을 통해서 기존 DataSet에 결과와 함께 append 한다

```
143  Show_KNN(3,"E")
144  Show_KNN(3,"M")
145  Show_KNN(5,"E")
146  Show_KNN(5,"M")
147
```

line143~146는 실제로 KNN을 실행하는 구현부이다(main)

line143부터 차례대로 3-NN(Eucildean), 3-NN(Manhattan), 5-NN(Eucildean), 5-NN(Manhattan) 순으로 실행한다.

3. 실험(데이터)

실험 전 초기 DataSet (20개)

```
DataSet=[  #초기 DataSet
[2.50, 6.00, 0],[3.75, 8.00, 0],[2.25, 5.50, 0],[3.25, 8.25, 0],
[2.75, 7.50, 0],[4.50, 5.00, 0],[3.50, 5.25, 0],[3.00, 3.25, 0],
[4.00, 4.00, 0],[4.25, 3.75, 0],[2.00, 2.00, 0],[5.00, 2.50, 0],
[8.25, 8.50, 0],[5.75, 8.75, 1],[4.75, 6.25, 1],[5.50, 6.75, 1],
[5.25, 9.50, 1],[7.00, 4.25, 1],[7.50, 8.0, 1],[7.25, 5.75, 1]
]
```

실험 후 DataSet (25개)

4개의 verision으로 나뉘어지기 때문에 각각 실행시 마다 DataSet을 출력하도록 하여 확인하였음

이때 line140의 주석처리된 코드를 주석해제 후 실행하면 확인가능

3-NN (Eucildean) → 25개

```
3 - NN(with E ) classfications's DataSet is  [[2.5, 6.0, 0], [3.75, 8.0, 0], [2.25, 5.5, 0], [3.25, 8.25, 0],
[2.75, 7.5, 0], [4.5, 5.0, 0], [3.5, 5.25, 0], [3.0, 3.25, 0], [4.0, 4.0, 0], [4.25, 3.75, 0], [2.0, 2.0, 0],
[5.0, 2.5, 0], [8.25, 8.5, 0], [5.75, 8.75, 1], [4.75, 6.25, 1], [5.5, 6.75, 1], [5.25, 9.5, 1], [7.0, 4.25,
1], [7.5, 8.0, 1], [7.25, 5.75, 1], [6.75, 3, 0], [5.34, 6.0, 1], [4.67, 8.4, 1], [7.0, 7.0, 1], [7.8, 5.4,
1]]
```

3-NN (Manhattan) → 25개

```
3 - NN(with M ) classfications's DataSet is  [[2.5, 6.0, 0], [3.75, 8.0, 0], [2.25, 5.5, 0], [3.25, 8.25, 0],
[2.75, 7.5, 0], [4.5, 5.0, 0], [3.5, 5.25, 0], [3.0, 3.25, 0], [4.0, 4.0, 0], [4.25, 3.75, 0], [2.0, 2.0, 0],
[5.0, 2.5, 0], [8.25, 8.5, 0], [5.75, 8.75, 1], [4.75, 6.25, 1], [5.5, 6.75, 1], [5.25, 9.5, 1], [7.0, 4.25,
1], [7.5, 8.0, 1], [7.25, 5.75, 1], [6.75, 3, 0], [5.34, 6.0, 1], [4.67, 8.4, 0], [7.0, 7.0, 1], [7.8, 5.4,
1]]
```

5-NN (Eucildean) → 25개

```
5 - NN(with E ) classfications's DataSet is  [[2.5, 6.0, 0], [3.75, 8.0, 0], [2.25, 5.5, 0], [3.25, 8.25, 0],
[2.75, 7.5, 0], [4.5, 5.0, 0], [3.5, 5.25, 0], [3.0, 3.25, 0], [4.0, 4.0, 0], [4.25, 3.75, 0], [2.0, 2.0, 0],
[5.0, 2.5, 0], [8.25, 8.5, 0], [5.75, 8.75, 1], [4.75, 6.25, 1], [5.5, 6.75, 1], [5.25, 9.5, 1], [7.0, 4.25,
1], [7.5, 8.0, 1], [7.25, 5.75, 1], [6.75, 3, 0], [5.34, 6.0, 1], [4.67, 8.4, 1], [7.0, 7.0, 1], [7.8, 5.4,
1]]
```

5-NN (Manhattan) → 25개

```
5 - NN(with M ) classfications's DataSet is  [[2.5, 6.0, 0], [3.75, 8.0, 0], [2.25, 5.5, 0], [3.25, 8.25, 0],
[2.75, 7.5, 0], [4.5, 5.0, 0], [3.5, 5.25, 0], [3.0, 3.25, 0], [4.0, 4.0, 0], [4.25, 3.75, 0], [2.0, 2.0, 0],
[5.0, 2.5, 0], [8.25, 8.5, 0], [5.75, 8.75, 1], [4.75, 6.25, 1], [5.5, 6.75, 1], [5.25, 9.5, 1], [7.0, 4.25,
1], [7.5, 8.0, 1], [7.25, 5.75, 1], [6.75, 3, 0], [5.34, 6.0, 1], [4.67, 8.4, 1], [7.0, 7.0, 1], [7.8, 5.4,
1]]
```

초기 DataSet에는 총 20개의 Data만 입력 되어있었지만 5개의 Input Data를 classification하고 그 결과를 DataSet에 추가하여 총 25개의 DataSet이 완성

3. 실험(실험내용)

```
133    Show_KNN(3,"E")
134    Show_KNN(3,"M")
135    Show_KNN(5,"E")
136    Show_KNN(5,"M")
137
```

과제에서 주어진 requirement들로 K-Nearest Neighbor 실행 후 Voronoi tessellation 작성

4. 결과(3NN-Eucildean)

Data (6.75,3.0), (5.34,6.0), (4.67,8.4), (7.0,7.0), (7.8,5.4)을 3-NN(Eucildean)으로 classification

```
3 - Nearest Neighbor distance And Result (Use of Eucilidean distance)
[1.2747548783981961, 1]
[1.8200274723201295, 0]
[2.6100766272276377, 0]
[6.75, 3] 's classification is No(False)

3 - Nearest Neighbor distance And Result (Use of Eucilidean distance)
[0.6407807737440316, 1]
[0.7668767828015137, 1]
[1.3059862173851606, 0]
[5.34, 6.0] 's classification is Yes(True)

3 - Nearest Neighbor distance And Result (Use of Eucilidean distance)
[1.0031948963187562, 0]
[1.1352973178863763, 1]
[1.2435433245367848, 1]
[4.67, 8.4] 's classification is Yes(True)

3 - Nearest Neighbor distance And Result (Use of Eucilidean distance)
[1.118033988749895, 1]
[1.2747548783981961, 1]
[1.5206906325745548, 1]
[7.0, 7.0] 's classification is Yes(True)

3 - Nearest Neighbor distance And Result (Use of Eucilidean distance)
[0.6519202405202645, 1]
[1.4008925726121901, 1]
[1.7888543819998313, 1]
[7.8, 5.4] 's classification is Yes(True)
```

(6.75, 3) 's classification : No

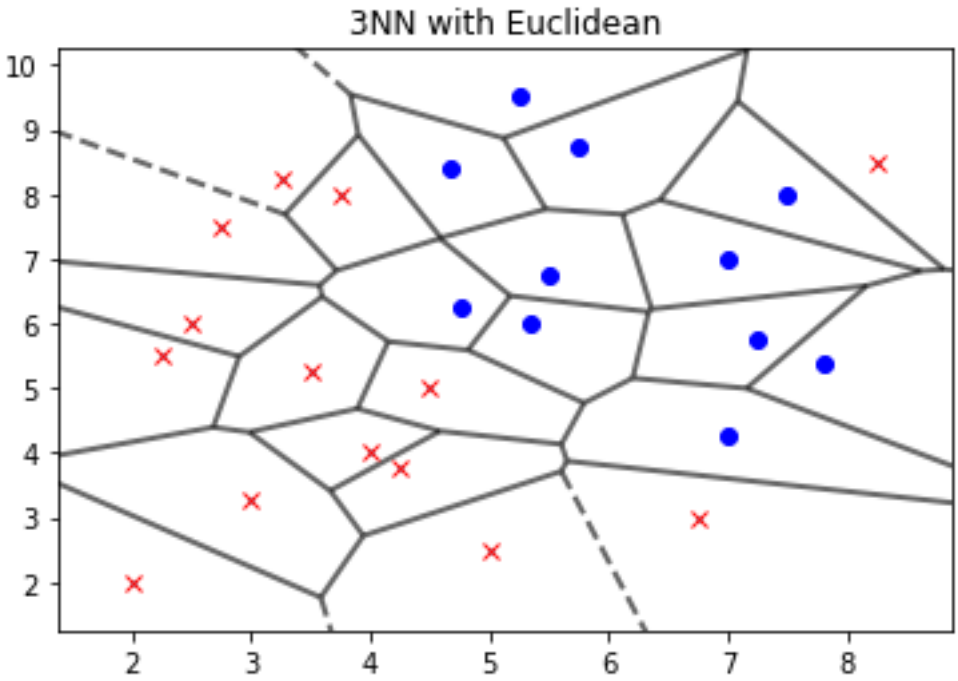
(5.34,6.0) 's classification : Yes

(4.67,8.4) 's classification : Yes

(7.0,7.0) 's classification : Yes

(7.8,5.4) 's classification : Yes

4. 결과(3NN-Eucildean Vornoi tessellation)



K-Nearest Neighbor 으로 classification한 이후 각각의 Data들이 어떠한 분포를 이루는지 확인하기 위해 Vornoi tessellation을 출력해보았다. 문제에서 제시한 5개의 Data (6.75,3.0), (5.34,6.0), (4.67,8.4), (7.0,7.0), (7.8,5.4) 가 실제로 classification되어 0,1,1,1,1을 가진 채로 적절한 위치에 위치해 있는 것을 확인할 수 있었다.

4. 결과(3NN-Manhattan)

Data (6.75,3.0), (5.34,6.0), (4.67,8.4), (7.0,7.0), (7.8,5.4)을 3-NN(Manhattan)으로 classification

```
 3 - Nearest Neighbor distance And Result (Use of Manhattan distance)
[1.5, 1]
[2.25, 0]
[3.25, 0]
[6.75, 3] 's classification is No(False)

3 - Nearest Neighbor distance And Result (Use of Manhattan distance)
[0.8399999999999999, 1]
[0.9100000000000001, 1]
[1.8399999999999999, 0]
[5.34, 6.0] 's classification is Yes(True)

3 - Nearest Neighbor distance And Result (Use of Manhattan distance)
[1.3200000000000003, 0]
[1.4299999999999997, 1]
[1.5700000000000003, 0]
[4.67, 8.4] 's classification is No(False)

3 - Nearest Neighbor distance And Result (Use of Manhattan distance)
[1.5, 1]
[1.5, 1]
[1.75, 1]
[7.0, 7.0] 's classification is Yes(True)

3 - Nearest Neighbor distance And Result (Use of Manhattan distance)
[0.8999999999999995, 1]
[1.9500000000000002, 1]
[2.3999999999999995, 1]
[7.8, 5.4] 's classification is Yes(True)
```

(6.75, 3) 's classification : No

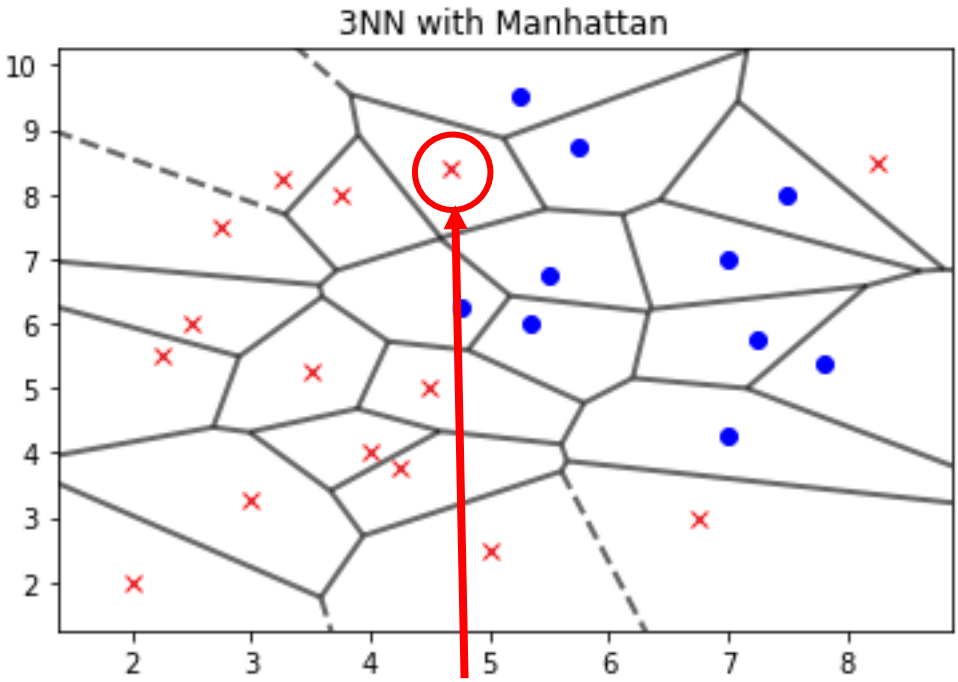
(5.34,6.0) 's classification : Yes

(4.67,8.4) 's classification : No

(7.0,7.0) 's classification : Yes

(7.8,5.4) 's classification : Yes

4. 결과(3NN-Manhattan Vornoi tessellation)



K-Nearest Neighbor 으로 classification한 이후 각각의 Data들이 어떠한 분포를 이루는지 확인하기 위해 Vornoi tessellation을 출력해보았다. 문제에서 제시한 5개의 Data (6.75,3.0), (5.34,6.0), (4.67,8.4), (7.0,7.0), (7.8,5.4) 가 실제로 classification되어 0,1,0,1,1을 가진 채로 적절한 위치에 위치해 있는 것을 확인할 수 있었다.

4. 결과(5NN-Eucilean)

Data (6.75,3.0), (5.34,6.0), (4.67,8.4), (7.0,7.0), (7.8,5.4)을 5-NN(Eucildean)으로 classification

```
5 - Nearest Neighbor distance And Result (Use of Eucilidean distance)
[1.2747548783981961, 1]
[1.8200274723201295, 0]
[2.6100766272276377, 0]
[2.7950849718747373, 1]
[2.9261749776799064, 0]
[6.75, 3] 's classification is No(False)

5 - Nearest Neighbor distance And Result (Use of Eucilidean distance)
[0.6407807737440316, 1]
[0.7668767828015137, 1]
[1.3059862173851606, 0]
[1.9262917743685666, 1]
[1.986982637065558, 0]
[5.34, 6.0] 's classification is Yes(True)

5 - Nearest Neighbor distance And Result (Use of Eucilidean distance)
[1.0031948963187562, 0]
[1.1352973178863763, 1]
[1.2435433245367848, 1]
[1.4279005567615695, 0]
[1.846997563615069, 1]
[4.67, 8.4] 's classification is Yes(True)

5 - Nearest Neighbor distance And Result (Use of Eucilidean distance)
[1.118033988749895, 1]
[1.2747548783981961, 1]
[1.5206906325745548, 1]
[1.9379370474811612, 1]
[1.9525624189766635, 0]
[7.0, 7.0] 's classification is Yes(True)

5 - Nearest Neighbor distance And Result (Use of Eucilidean distance)
[0.6519202405202645, 1]
[1.4008925726121901, 1]
[1.7888543819998313, 1]
[2.532113741521103, 1]
[2.61725046566048, 1]
[7.8, 5.4] 's classification is Yes(True)
```

(6.75, 3) 's classification : No

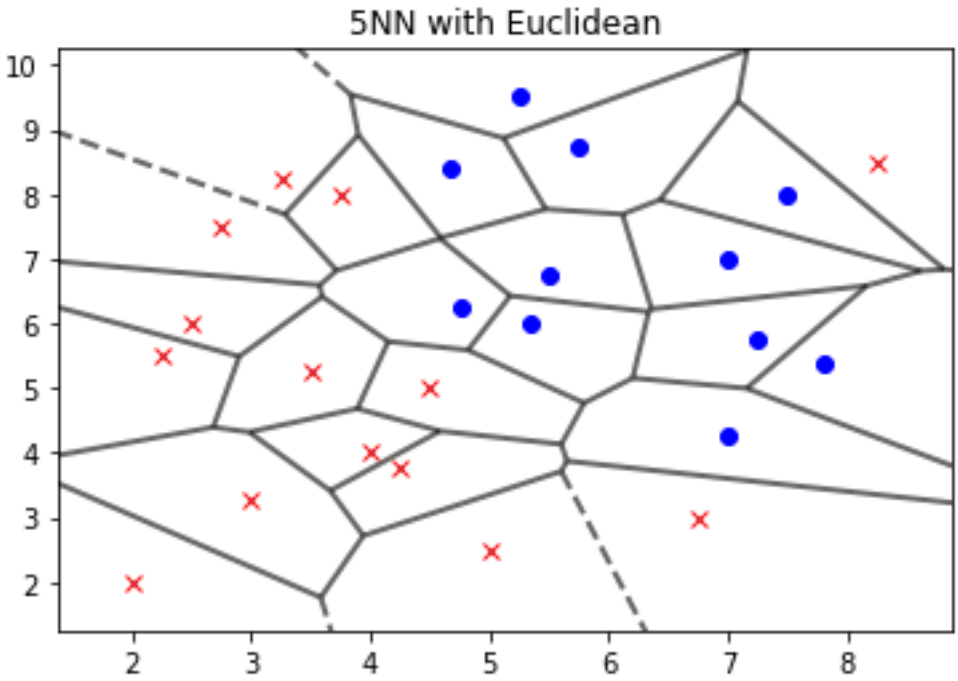
(5.34,6.0) 's classification : Yes

(4.67,8.4) 's classification : Yes

(7.0,7.0) 's classification : Yes

(7.8,5.4) 's classification : Yes

4. 결과(5NN-Eucildean Vornoi tessellation)



K-Nearest Neighbor 으로 classification한 이후 각각의 Data들이 어떠한 분포를 이루는지 확인하기 위해 Vornoi tessellation을 출력해보았다. 문제에서 제시한 5개의 Data (6.75,3.0), (5.34,6.0), (4.67,8.4), (7.0,7.0), (7.8,5.4) 가 실제로 classification되어 0,1,1,1,1을 가진 채로 적절한 위치에 위치해 있는 것을 확인할 수 있었다.

4. 결과(5NN-Manhattan)

Data (6.75,3.0), (5.34,6.0), (4.67,8.4), (7.0,7.0), (7.8,5.4)을 5-NN(Manhattan)으로 classification

```
5 - Nearest Neighbor distance And Result (Use of Manhattan distance)
[1.5, 1]
[2.25, 0]
[3.25, 0]
[3.25, 1]
[3.75, 0]
[6.75, 3] 's classification is No(False)

5 - Nearest Neighbor distance And Result (Use of Manhattan distance)
[0.8399999999999999, 1]
[0.9100000000000001, 1]
[1.8399999999999999, 0]
[2.16, 1]
[2.59, 0]
[5.34, 6.0] 's classification is Yes(True)

5 - Nearest Neighbor distance And Result (Use of Manhattan distance)
[1.3200000000000003, 0]
[1.4299999999999997, 1]
[1.5700000000000003, 0]
[1.6799999999999997, 1]
[2.2300000000000004, 1]
[4.67, 8.4] 's classification is Yes(True)

5 - Nearest Neighbor distance And Result (Use of Manhattan distance)
[1.5, 1]
[1.5, 1]
[1.75, 1]
[2.66, 1]
[2.75, 0]
[7.0, 7.0] 's classification is Yes(True)

5 - Nearest Neighbor distance And Result (Use of Manhattan distance)
[0.8999999999999995, 1]
[1.9500000000000002, 1]
[2.3999999999999995, 1]
[2.8999999999999995, 1]
[3.0599999999999996, 1]
[7.8, 5.4] 's classification is Yes(True)
```

(6.75, 3) 's classification : No

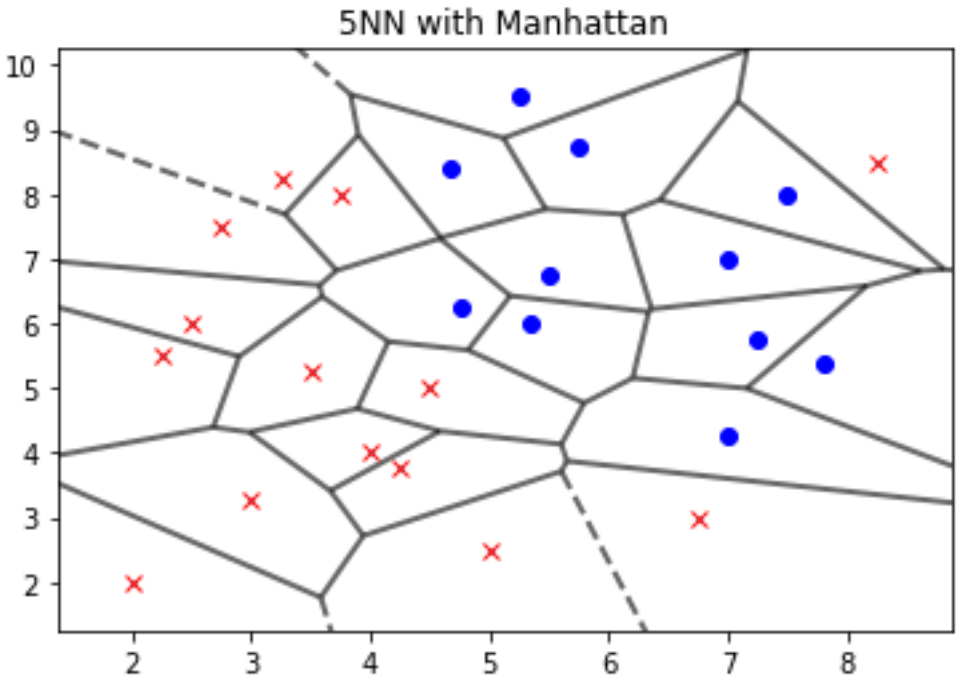
(5.34,6.0) 's classification : Yes

(4.67,8.4) 's classification : Yes

(7.0,7.0) 's classification : Yes

(7.8,5.4) 's classification : Yes

4. 결과(5NN-Manhattan Vornoi tessellation)



K-Nearest Neighbor 으로 classification한 이후 각각의 Data들이 어떠한 분포를 이루는지 확인하기 위해 Vornoi tessellation을 출력해보았다. 문제에서 제시한 5개의 Data (6.75,3.0), (5.34,6.0), (4.67,8.4), (7.0,7.0), (7.8,5.4) 가 실제로 classification되어 0,1,1,1,1을 가진 채로 적절한 위치에 위치해 있는 것을 확인할 수 있었다.

5. 결론 및 고찰

4. 결과 들을 보면 KNN을 사용하여 결과를 classification하였을때의 출력과 Vornoi tessellation을 통한 출력을 확인할 수 있다. 과제에서 제시한 5개의 추가 Input Data가 제대로 총 DataSet에 포함되었는지를 확인하기위해 추가코드를 작성하였다. 이는 보고서와 함께 제출된 코드의 line130을 주석해제 후 실행하면 각 버전의 KNN으로 classification 한 뒤에 전체 DataSet의 요소를 출력한다. 이때 가장 마지막으로 들어간 5번째 Data까지 모두 DataSet에 추가된 것을 확인할 수 있다.

4. 결과에 있는 Vornoi tessellation을 확인하면 각각의 KNN으로 classification 한 결과를 분석하면 모두 같은 위치에 있는것으로 Vornoi tessellation이 그려진다. 하지만 이때의 classification 결과가 3-NN(E), 5-NN(E), 5-NN(M)은 모두 같지만 3-NN(M)의 경우 Input Data(4.67, 8.4)의 classification 결과가 0으로 혼자 다른 것을 확인할 수 있다. 이를 통해 KNN방법을 사용시 K의 값에 따라서 (이때 K는 홀수만을 사용) classification 결과가 달라질 수 있음을 인지하여 최적의 K값을 구해야만 정확도가 상승함을 알게되었다.

KNN은 거리만을 측정하여 classification 하기 때문에 단순하고 효율적이다. 또한 한번 classification 한 결과값을 DataSet에 추가할 수 있기때문에 학습에 도움을 줄 수 있다는 장점도 있다(DataSet의 크기가 클수록 정확도는 상승). 하지만 모델을 생성하는 것이 아니기 때문에 feature와 label간의 관계를 설명하기에는 다소 어려움이 있다.(Input Data (4.67, 8.4)의 경우 참고) 때문에 최적의 K값을 찾는 것과 distance 측정방법을 정하는 것 이 가장 중요하다.

이번 과제를 진행하면서 KNN을 이론뿐만이 아니라 직접 구현해보면서 정확히 어떤 방식으로 작동되는지 알 수 있는 좋은 기회가 되었다. 또한 google doc에 올라온 질문 중 class별로 다르게 나오는 표기에 대한 질문이 있어 scatter plot에 대해 공부하고 여러 시도 끝에 성공하였다. 과정속에서 scatter plot에 대해 더욱 알아볼 수 있는 기회가 된 것 같아 만족스럽다.