목차

휴먼지능정보공학과 201910792 김종한

1. 접근

4개 버전에서 판별하는 'Draft'값들이 대부분 비슷할 것이라고 예상한다. 이유는 거리 계산 함수가 달라도 최근접 값은 거의 동일하게 선정될 것이라고 생각한다. 따라서 각 버전마다 'Draft'값 또한 같게 나올 것이라 예상된다.

각 버전의 Voronoi 그래프를 그리기 위해서 필요한, 유클리드 함수와 맨허튼 함수를 구현하고, 최근접 값을 구하는 함수, 'Draft'값을 선별하여 반환하는 함수, 새로운 데이터를 추가하고 모델을 업데이트 하는 함수를 구현한다.

2. 구현

1) dataset과 testset 값 설정

20개의 samples를 dataset 리스트로, 주어진 5개의 데이터를 testset 리스트로 만들어준다.

ID	Speed	Agility	Draft	ID	Speed	Agility	Draft
1	2.50	6.00	No	11	2.00	2.00	No
2	3.75	8.00	No	12	5.00	2.50	No
3	2.25	5.50	No	13	8.25	8.50	No
4	3.25	8.25	No	14	5.75	8.75	Yes
5	2.75	7.50	No	15	4.75	6.25	Yes
6	4.50	5.00	No	16	5.50	6.75	Yes
7	3.50	5.25	No	17	5.25	9.50	Yes
8	3.00	3.25	No	18	7.00	4.25	Yes
9	4.00	4.00	No	19	7.50	8.00	Yes
10	4.25	3.75	No	20	7.25	5.75	Yes

(6.75, 3), (5.34, 6.0), (4, 67, 8.4), (7.0, 7.0), (7.8, 5, 4)

```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# [Speed, Agility, Draft]
dataset=[[2.50, 6.00, 'No'],
         [3.75, 8.00, 'No'],
         [2.25, 5.50, 'No'],
         [3.25, 8.25, 'No'],
         [2.75, 7.50, 'No'],
         [4.50, 5.00, 'No'],
         [3.50, 5.25, 'No'],
         [3.00, 3.25, 'No'],
         [4.00, 4.00, 'No'],
         [4.25, 3.75, 'No'],
         [2.00, 2.00, 'No'],
         [5.00, 2.50, 'No'],
         [8.25, 8.50, 'No'],
         [5.75, 8.75, 'Yes'],
         [4.75, 6.25, 'Yes'],
         [5.50, 6.75, 'Yes'],
         [5.25, 9.50, 'Yes'],
         [7.00, 4.25, 'Yes'],
         [7.50, 8.00, 'Yes'],
         [7.25, 5.75, 'Yes']]

testset = [[6.75,3],[5.34,6.0],[4.67,8.4],[7.0,7.0],[7.8,5.4]]
```

2) Euclidean 함수와 Manhattan 함수 구현

다음은 Euclidean distance 함수를 구현한 것이다.
강의 시간에 배운 Euclidean 수식을 토대로 구현했다.

$$Euclidean(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{m} (\mathbf{a}[i] - \mathbf{b}[i])^2}$$

```python
# 유클리드 거리 함수

def euclidean_distance(row1,row2):
    distance = 0
    for i in range(len(row1)-1):
        distance += (row1[i]-row2[i])**2
    return distance**0.5
```

$$Manhattan(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{m} abs(\mathbf{a}[i] - \mathbf{b}[i])$$

```python
# 맨허튼 거리 함수

def manhattan_distance(row1,row2):
    distance = 0
    for i in range(len(row1)-1):
        distance += abs(row1[i]-row2[i])
    return distance
```

row1값은 dataset의 값들을 의미한다.
row1은 Speed, Agility, Draft값을 모두 가지고 있으므로, 거리를 계산하기 위해서 숫자값인 Speed와 Agility값들로 계산하는 과정을 거친다.

두 함수의 반복문에서 (len(row1)-1)을 한 이유는 row1에는 각 데이터의 'Draft'값이 포함되어 있는데 각 데이터 간의 거리를 구할 땐 'Draft'값은 포함하지 않기 때문에 –1을 해준 것이다.

3) Euclidean distance를 사용하여 최근접 값을 구하는 함수

앞으로 보여주는 함수는 모두 Euclidean distance를 사용한 함수들이다. Manhattan distance를 사용한 함수들은 모두 형식은 같고, Euclidean distance가 사용된 자리에 Manhattan distance를 넣으면 된다.

다음은 Euclidean distance 함수를 사용하여 test_row와 거리가 가까운 3개의 train_row를 선별하는 함수이다.

```
# 3-Nearest neighbor, 유클리드 거리 함수 사용버전

def _3NN_Euclidean(train,test_row):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(train_row,test_row)
        distances.append((train_row,dist))
    distances.sort(key=lambda tup :tup[1])
    neighbors = list()
    for i in range(3):
        neighbors.append(distances[i][0])
    return neighbors
```

20개의 sample자료들과 추가되는 데이터들 간의 거리를 구하고, 거리값들을 distances 리스트에 넣는다. lambda를 이용해서 오름차순으로 정렬한다.
neighbors 리스트에 distances 리스트에서 가장 작은 값 세 개를 순서대로 넣는다. 만약 k가 5라면 반복문 조건에 3대신에 5를 입력하면 된다. neighbors 리스트를 반환한다.

4) 'Draft'값 반환 함수

다음은 test값들의 'Draft' 값을 예측하여 반환하는 함수이다.

```
# _3NN_Euclidean 예측 함수

def predict_classification_3NN_Euclidean(train, test_row):
    neighbors = _3NN_Euclidean(train, test_row)
    for neighbor in neighbors:
        print(neighbor)
    output_values = list()
    for row in neighbors:
        output_values.append(row[2])
    prediction = max(set(output_values), key=output_values.count) # output_values에
    return prediction
```

앞의 최근접 값들을 반환하는 함수를 이용하여 얻었던 3개의 가까운 값들을 출력하여 보여준다.

그 3개 값들의 'Draft'를 output_values 리스트에 모으고, 리스트 안에서 가장 수가 많은 'Draft' 값(ex. 'Yes' or 'No')을 prediction에 넣고 반환한다.

5) 새로운 데이터를 추가하고, 모델을 업데이트

다음은 'Draft'값을 판별한 새로운 값들을 dataset에 추가하는 과정이다.

```
list_in = list()

for test_row in range(len(testset)):
    prediction = predict_classification_3NN_Euclidean(dataset, testset[test_row])
    print('Expected: %s' % (prediction))
    print(sep = " ")

    list_in.append(testset[test_row][0])
    list_in.append(testset[test_row][1])
    list_in.append(str(prediction))
    dataset.append(list_in)

    list_in=list()
```

list_in 리스트를 만들어서 추가되는 데이터의 'Speed' , 'Agility', 'Draft' 값들을 list_in 리스트에 넣고, list_in을 dataset에 넣어서 새로운 데이터를 추가하여 dataset을 업데이트 시킨다.
그 결과, 그 다음에 추가되는 데이터는 이전에 추가된 값까지 고려할 수 있게 된다.
다음은 위의 코드를 실행한 결과이다. 결과를 해석해보자.

```
[7.0, 4.25, 'Yes']
[5.0, 2.5, 'No']
[4.25, 3.75, 'No']
Expected: No

[4.75, 6.25, 'Yes']
[5.5, 6.75, 'Yes']
[4.5, 5.0, 'No']
Expected: Yes

[3.75, 8.0, 'No']
[5.75, 8.75, 'Yes']
[5.25, 9.5, 'Yes']
Expected: Yes

[7.5, 8.0, 'Yes']
[7.25, 5.75, 'Yes']
[5.5, 6.75, 'Yes']
Expected: Yes

[7.25, 5.75, 'Yes']
[7.0, 4.25, 'Yes']
[7.0, 7.0, 'Yes']
Expected: Yes
```

[6.75, 3]와 가장 가까운 순서대로 세 개의 값들이 출력되었다.
값의 'Draft' 수가 'Yes'는 1개, 'No'는 2개 이므로 [6.75, 3] 의 'Draft' 값은 'No'가 된다.

[5.34, 6.0]와 가장 가까운 순서대로 세 개의 값들이 출력되었다.
값의 'Draft' 수가 'Yes'는 2개, 'No'는 1개 이므로 [5.34, 6.0] 의 'Draft' 값은 'Yes'가 된다.

[4.67, 8.4]와 가장 가까운 순서대로 세 개의 값들이 출력되었다.
값의 'Draft' 수가 'Yes'는 2개, 'No'는 1개 이므로 [6.75, 3] 의 'Draft' 값은 'Yes'가 된다.

[7.0, 7.0]와 가장 가까운 순서대로 세 개의 값들이 출력되었다.
값의 'Draft' 수가 'Yes'는 3개, 'No'는 0개 이므로 [7.0, 7.0] 의 'Draft' 값은 'Yes'가 된다.

앞서 추가된 데이터인 [7.0,7.0,'Yes']도 업데이트 되어 그 다음에 추가되는 데이터가 이 값까지 고려할 수 있게 됐다는 것을 볼 수 있다.

[7.8, 5.4]와 가장 가까운 순서대로 세 개의 값들이 출력되었다.
값의 'Draft' 수가 'Yes'는 3개, 'No'는 0개 이므로 [7.8, 5.4] 의 'Draft' 값은 'Yes'가 된다.

6) 최근접 값들이 잘 선정됐는지 Test

다음은 최근접 값들이 잘 선정됐는지 확인해보기 위한 코드이다.
dataset의 row와 추가되는 row0 간의 거리를 모두 출력한다.

```
#  Test

dist_list = list()
row0 = [6.75, 3]
for row in dataset:
    distance = euclidean_distance(row,row0)
    dist_list.append(distance)
    print(distance)
min(dist_list)
```

5.202163011671203
5.830951894845301
5.1478150704935
6.309714732061981
6.020797289396148
3.010398644698074
3.952847075210474
3.758324094593227
2.9261749776799064
2.6100766272276377 3등
4.8541219597369
1.8200274723201295 2등
5.70087712549569
5.836308764964376
3.816084380618437
3.952847075210474
6.670832032063167
1.2747548783981961 1등
5.055937104039171
2.7950849718747373

ID	Speed	Agility	Draft	ID	Speed	Agility	Draft
1	2.50	6.00	No	11	2.00	2.00	No
2	3.75	8.00	No	12	5.00	2.50	No
3	2.25	5.50	No	13	8.25	8.50	No
4	3.25	8.25	No	14	5.75	8.75	Yes
5	2.75	7.50	No	15	4.75	6.25	Yes
6	4.50	5.00	No	16	5.50	6.75	Yes
7	3.50	5.25	No	17	5.25	9.50	Yes
8	3.00	3.25	No	18	7.00	4.25	Yes
9	4.00	4.00	No	19	7.50	8.00	Yes
10	4.25	3.75	No	20	7.25	5.75	Yes

[6.75, 3]과
가장 거리가 가까운 값 1등의 ID는 18이고, 그 값은 [7.0, 4.25, 'Yes'] 이다.

2등의 ID는 12이고,
그 값은 [5.00, 2.50, 'No'] 이다.

3등의 ID는 10이고,
그 값은 [4.25, 3.75, 'No'] 이다.

이 값들은 위에서 선정된 최근접 값 3개와 동일한 값임을 볼 수 있다.

```
[7.0, 4.25, 'Yes']
[5.0, 2.5, 'No']
[4.25, 3.75, 'No']
Expected: No
```

7) DataFrame 생성

Voronoi 그래프를 그리기 위해서 dataset(기존 데이터 20개 + 추가된 5개 데이터 = 총 25개의 데이터)를 DataFrame을 만든다.

다음은 list를 이용해서 데이터 프레임을 만드는 코드이다.
데이터 프레임 df는 'Speed', 'Agility', 'Draft' 세 열을 가진다.

```
list_speed=list()
for i in range (len(dataset)):
    list_speed.append(dataset[i][0])

list_agility=list()
for i in range (len(dataset)):
    list_agility.append(dataset[i][1])

list_draft=list()
for i in range (len(dataset)):
    list_draft.append(dataset[i][2])

data={'Speed': list_speed, 'Agility': list_agility, 'Draft': list_draft}
df = pd.DataFrame(data)
df
```

실행 결과는 다음과 같다.

	Speed	Agility	Draft
0	2.50	6.00	No
1	3.75	8.00	No
2	2.25	5.50	No
3	3.25	8.25	No
4	2.75	7.50	No
5	4.50	5.00	No
6	3.50	5.25	No
7	3.00	3.25	No
8	4.00	4.00	No
9	4.25	3.75	No
10	2.00	2.00	No
11	5.00	2.50	No
12	8.25	8.50	No
13	5.75	8.75	Yes
14	4.75	6.25	Yes
15	5.50	6.75	Yes
16	5.25	9.50	Yes
17	7.00	4.25	Yes
18	7.50	8.00	Yes
19	7.25	5.75	Yes
20	6.75	3.00	No
21	5.34	6.00	Yes
22	4.67	8.40	Yes
23	7.00	7.00	Yes
24	7.80	5.40	Yes

dataset에 있던 데이터들이
df 데이터 프레임으로 만들어진 것을 확인할 수 있다.

기존에 주어진 20개의 데이터들이다.

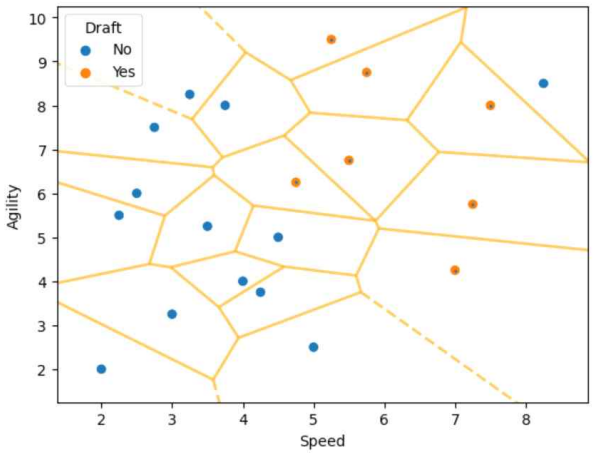
새로 추가된 5개의 데이터들이다.

3. 실험

1) 3-NN Euclidean distance Voronoi 그래프 그리기

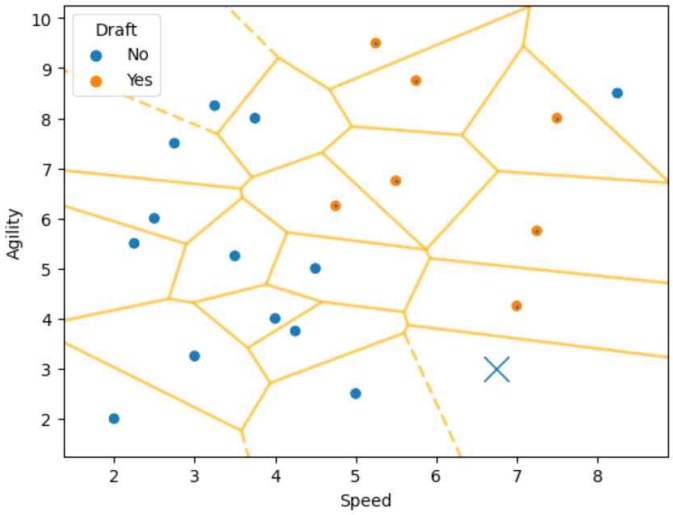
df를 loc함수를 사용하여 원하는 행들을 추출해서 그래프를 그려본다.

다음은 처음에 주어진 20개의 데이터만 존재하는 그래프이다.



아래 그림은 20개의 데이터가 존재하는 데이터 세트에서 [6.75, 3] 값이 추가된 그래프이다.

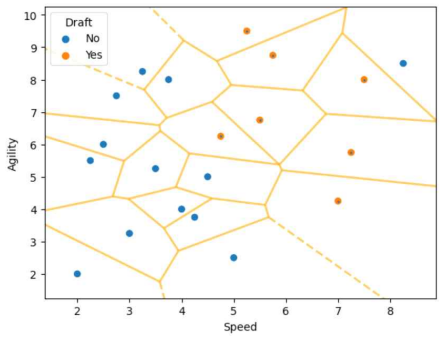
```
fig = voronoi_plot_2d(vor, show_vertices=False, line_colors='orange',line_width=2, line_alpha=0.6, point_size=2)
fig = sns.scatterplot(data = c, x='Speed', y='Agility', hue='Draft', s=50)
fig = sns.scatterplot(data = d, x='Speed', y='Agility', hue='Draft',marker='x', s=220, legend=False)
plt.show()
```



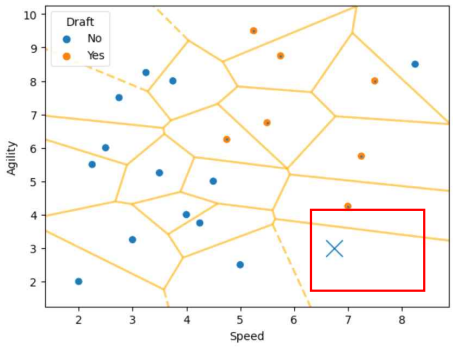
추가된 값은 X모양으로 표시해줬다.

새로운 값이 추가될 때마다 어떻게 그래프가 변하는지 비교해보자.

<처음 20개의 데이터로 그린 그래프>

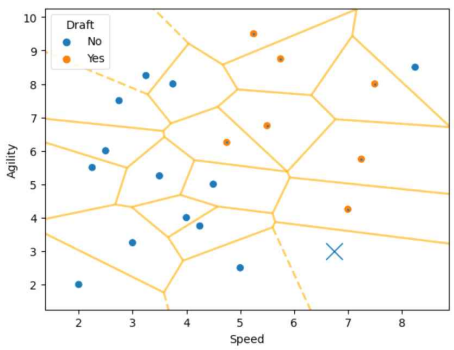


<[6.75,3]값이 추가된 그래프>

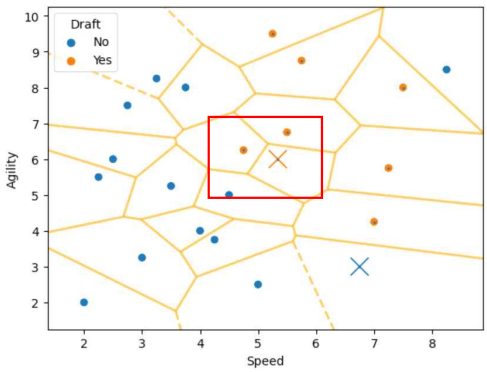


[6.75, 3]의 'Draft' 값인 'No'가 들어갔고, 영역을 차지했음을 볼 수 있다.
그래프의 다른 부분은 전부 동일한 것을 볼 수 있지만, 값이 추가된 오른쪽 아래 부분이 [6.75, 3] 값의 추가를 반영하여 경계가 달라졌음을 볼 수 있다.

<데이터가 1개 추가된 그래프>

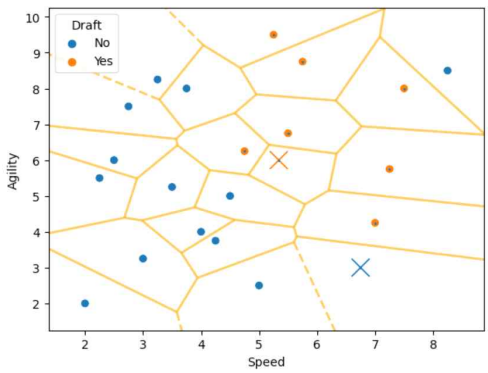


<데이터가 2개 추가된 그래프>

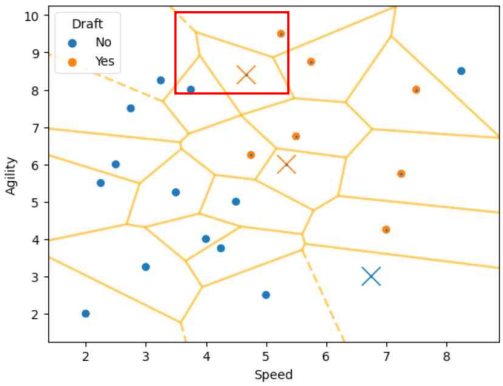


두 번째 순서값인 [5.34, 6.0]이 들어간 그래프와 비교해보자.
[5.34, 6.0]의 'Draft' 값인 'Yes'가 들어갔고, 영역을 차지했음을 볼 수 있다.
가운데 부분이 [5.34, 6.0] 값의 추가를 반영하여 경계가 달라졌음을 볼 수 있다.

<데이터가 2개 추가된 그래프>

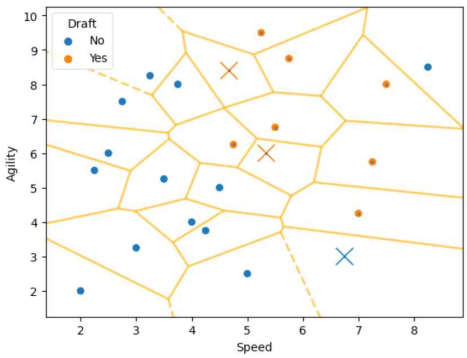


<데이터가 3개 추가된 그래프>

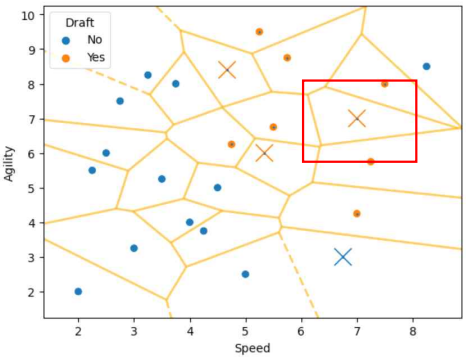


세 번째 순서값인 [4.67, 8.4]이 들어간 그래프와 비교해보자.
[4.67, 8.4]의 'Draft' 값인 'Yes'가 들어갔고, 영역을 차지했음을 볼 수 있다.
표시한 부분이 [4.67, 8.4] 값의 추가를 반영하여 경계가 달라졌음을 볼 수 있다.

<데이터가 3개 추가된 그래프>

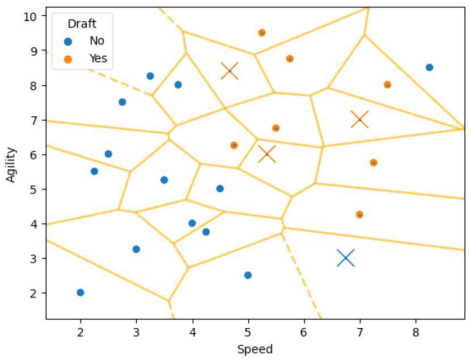


<데이터가 4개 추가된 그래프>

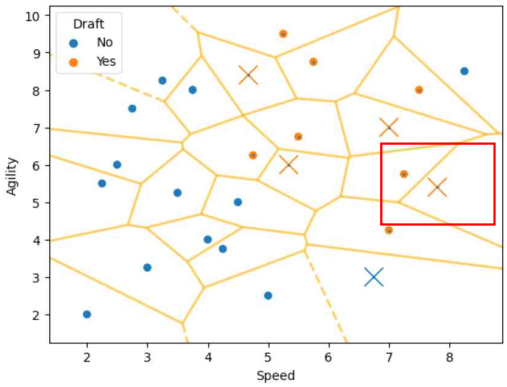


네 번째 순서값인 [7.0, 7.0]이 들어간 그래프와 비교해보자.
[7.0, 7.0]의 'Draft' 값인 'Yes'가 들어갔고, 영역을 차지했음을 볼 수 있다.
표시한 부분이 [7.0, 7.0] 값의 추가를 반영하여 경계가 달라졌음을 볼 수 있다.

<데이터가 4개 추가된 그래프>



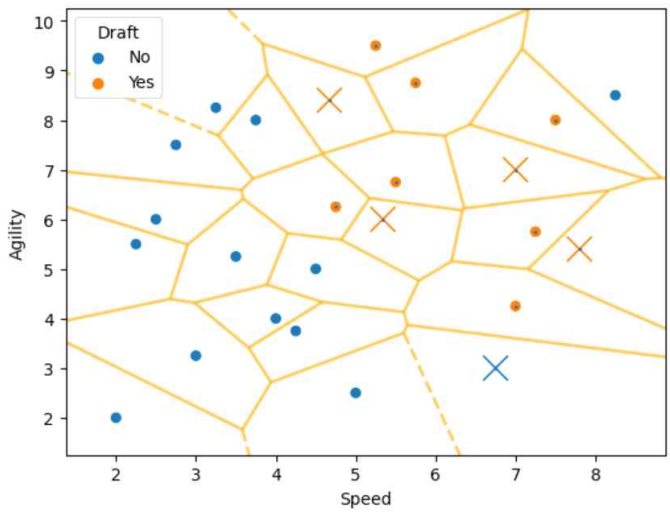
<데이터가 5개 추가된 그래프>



다섯 번째 순서값인 [7.8, 5.4]이 들어간 그래프와 비교해보자.
[7.8, 5.4]의 'Draft' 값인 'Yes'가 들어갔고, 영역을 차지했음을 볼 수 있다.
표시한 부분이 [7.8, 5.4] 값의 추가를 반영하여 경계가 달라졌음을 볼 수 있다.

따라서, 5개의 데이터를 모두 포함한 3-NN Euclidean버전의 그래프는 다음과 같다.

```
fig = voronoi_plot_2d(vor, show_vertices=False, line_colors='orange',line_width=2,
fig = sns.scatterplot(data = c, x='Speed', y='Agility', hue='Draft', s=50)
fig = sns.scatterplot(data = test, x='Speed', y='Agility', hue='Draft',marker='x',
plt.show()
```



2) 5-NN Euclidean distance

2-1) 3-NN Euclidean distance와 다른점

코드 부분에서는 큰 차이가 없다. 결과에서 차이점만 보자.

```
[7.0, 4.25, 'Yes']
[5.0, 2.5, 'No']
[4.25, 3.75, 'No']
[7.25, 5.75, 'Yes']
[4.0, 4.0, 'No']
Expected: No

[4.75, 6.25, 'Yes']
[5.5, 6.75, 'Yes']
[4.5, 5.0, 'No']
[7.25, 5.75, 'Yes']
[3.5, 5.25, 'No']
Expected: Yes

[3.75, 8.0, 'No']
[5.75, 8.75, 'Yes']
[5.25, 9.5, 'Yes']
[3.25, 8.25, 'No']
[5.5, 6.75, 'Yes']
Expected: Yes

[7.5, 8.0, 'Yes']
[7.25, 5.75, 'Yes']
[5.5, 6.75, 'Yes']
[5.34, 6.0, 'Yes']
[8.25, 8.5, 'No']
Expected: Yes

[7.25, 5.75, 'Yes']
[7.0, 4.25, 'Yes']
[7.0, 7.0, 'Yes']
[5.34, 6.0, 'Yes']
[7.5, 8.0, 'Yes']
Expected: Yes
```

5-NN Euclidean

```
[7.0, 4.25, 'Yes']
[5.0, 2.5, 'No']
[4.25, 3.75, 'No']
Expected: No

[4.75, 6.25, 'Yes']
[5.5, 6.75, 'Yes']
[4.5, 5.0, 'No']
Expected: Yes

[3.75, 8.0, 'No']
[5.75, 8.75, 'Yes']
[5.25, 9.5, 'Yes']
Expected: Yes

[7.5, 8.0, 'Yes']
[7.25, 5.75, 'Yes']
[5.5, 6.75, 'Yes']
Expected: Yes

[7.25, 5.75, 'Yes']
[7.0, 4.25, 'Yes']
[7.0, 7.0, 'Yes']
Expected: Yes
```

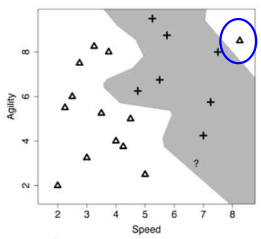
3-NN Euclidean

왼쪽은 k가 5, 오른쪽은 k가 3, 거리계산법은 Euclidean distance를 사용했을 때 얻은 최근접 값들과 'Draft'값이다.

k가 5이므로 최근접 값을 5개를 뽑았다.

k가 3인 경우의 최근접 값들을 포함하고, 그 다음으로 근접한 값 2개를 추가로 포함한다.

k가 3인 경우와 k가 5인 경우의 'Draft'값이 동일함을 볼 수 있다.

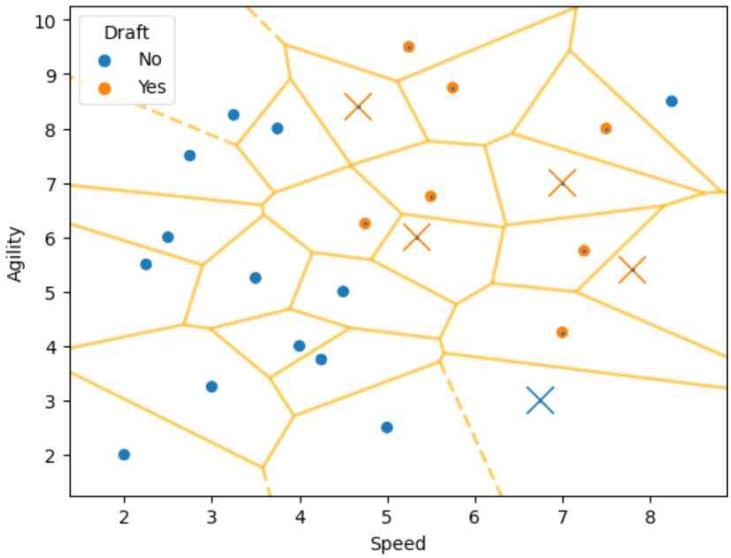


[8.25, 8.5, 'No']는 잘못 입력된 데이터 세트의 값인 결측값이라고 예상된다.
혹시 결측값에 의해 'Draft'값이 바뀌는지 봤지만, 다행히 결측값은 영향이 없었다.

2-2) Voronoi 그래프 그리기

다음은 5-NN Euclidean버전의 그래프이다.

```
fig = voronoi_plot_2d(vor, show_vertices=False, line_colors='orange', line_width=2, line_alpha=0.6, point_size=2)
fig = sns.scatterplot(data = c, x='Speed', y='Agility', hue='Draft', s=50)
fig = sns.scatterplot(data = d, x='Speed', y='Agility', hue='Draft', marker='x', s=220, legend=False)
plt.show()
```



3-NN Euclidean 버전과 동일한 모양임을 알 수 있다.
5-NN Euclidean 버전과 3-NN Euclidean 버전의 'Draft'값이 동일하므로 두 버전의 그래프 모양이 같게 나왔다. 또한, 새로운 값이 추가될 때 그래프 또한 모양이 같았다.

3) 3-NN Manhattan distance

3-1) 3-NN Euclidean distance와 다른점

코드 부분에서는 큰 차이가 없다. 결과에서 차이점만 보자.

```
[7.0, 4.25, 'Yes']
[5.0, 2.5, 'No']
[4.25, 3.75, 'No']
Expected: No

[4.75, 6.25, 'Yes']
[5.5, 6.75, 'Yes']
[4.5, 5.0, 'No']
Expected: Yes

[3.75, 8.0, 'No']
[5.75, 8.75, 'Yes']
[3.25, 8.25, 'No']
Expected: No

[7.5, 8.0, 'Yes']
[7.25, 5.75, 'Yes']
[5.5, 6.75, 'Yes']
Expected: Yes

[7.25, 5.75, 'Yes']
[7.0, 4.25, 'Yes']
[7.0, 7.0, 'Yes']
Expected: Yes
```

<3-NN Manhattan>

```
[7.0, 4.25, 'Yes']
[5.0, 2.5, 'No']
[4.25, 3.75, 'No']
Expected: No

[4.75, 6.25, 'Yes']
[5.5, 6.75, 'Yes']
[4.5, 5.0, 'No']
Expected: Yes

[3.75, 8.0, 'No']
[5.75, 8.75, 'Yes']
[5.25, 9.5, 'Yes']
Expected: Yes

[7.5, 8.0, 'Yes']
[7.25, 5.75, 'Yes']
[5.5, 6.75, 'Yes']
Expected: Yes

[7.25, 5.75, 'Yes']
[7.0, 4.25, 'Yes']
[7.0, 7.0, 'Yes']
Expected: Yes
```

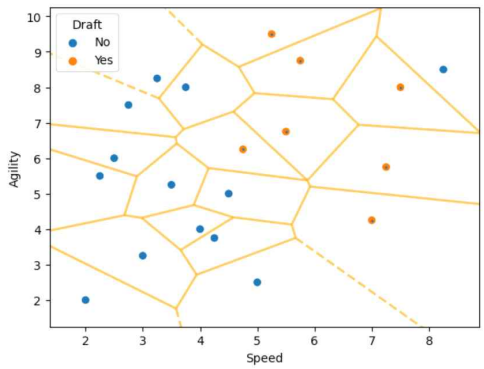
<3-NN Euclidean>

두 버전의 차이점이다. 세 번째로 추가되는 [4.67, 8.4]값은, Manhattan distance를 사용한 경우에는 3번째로 가까운 데이터가 [3.25, 8.25, 'No']이고, Euclidean distance를 사용한 경우에는 [5.25, 9.5, 'Yes']이다.
따라서 3-NN Manhattan에서는 [4.67, 8.4]데이터의 'Draft'는 'No'가 된다.

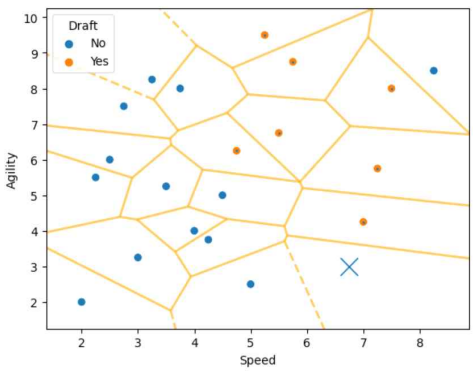
3-2) Voronoi 그래프 그리기

df를 loc함수를 사용하여 원하는 행들을 추출해서 그래프를 그려본다.

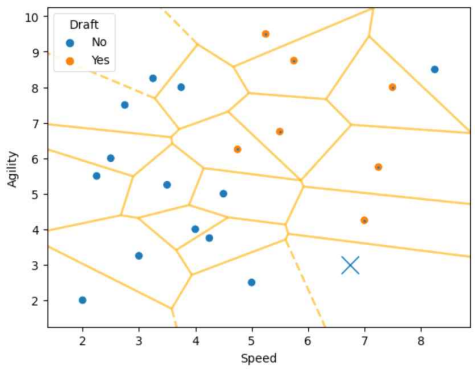
새로운 값이 추가될 때마다 어떻게 그래프가 변하는지 비교해보자.



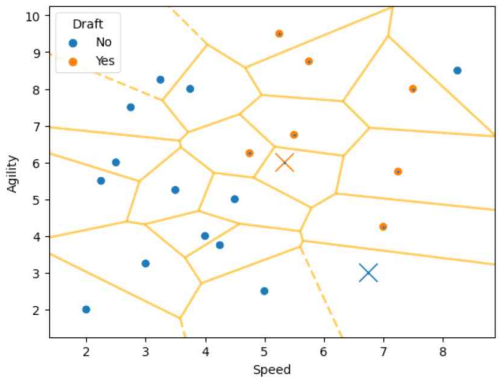
처음 20개의 데이터로 그린 그래프



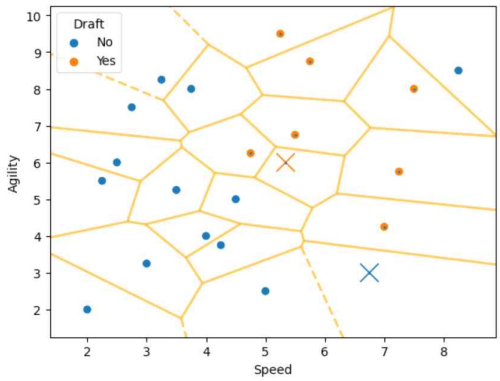
[6.75,3]값이 추가된 그래프



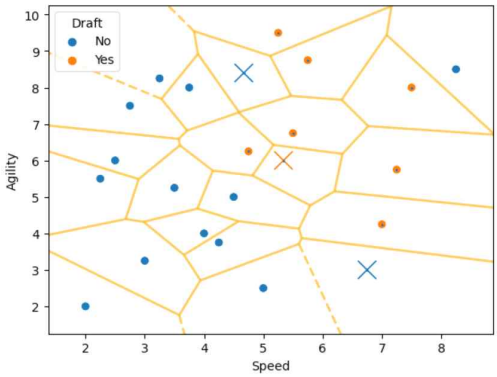
데이터가 1개 추가된 그래프



[5.34, 6.0]값이 추가된 그래프



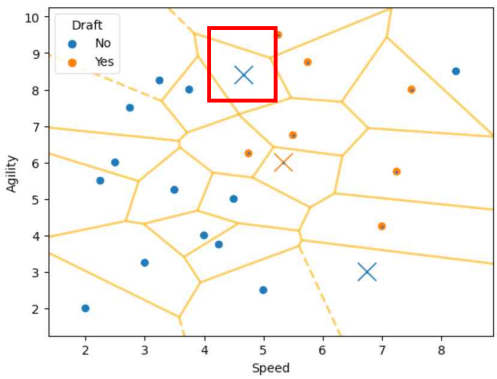
데이터가 2개 추가된 그래프



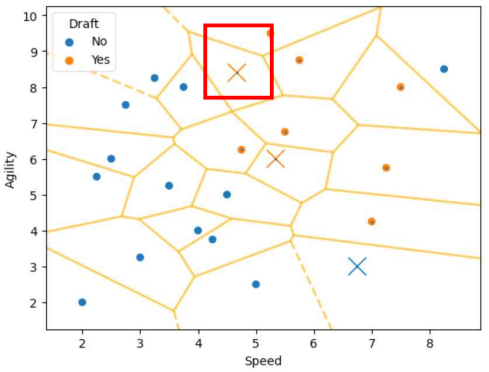
[4.67, 8.4]값이 추가된 그래프

세 번째로 추가되는 [4.67, 8.4]에서 3-NN Euclidean distance와 차이점이 나타난다.

다음은 [4.67, 8.4]값이 추가 될 때 3-NN Manhattan distance와 3-NN Euclidean distance의 voronoi그래프이다. 왼쪽이 Manhattan distance이다.

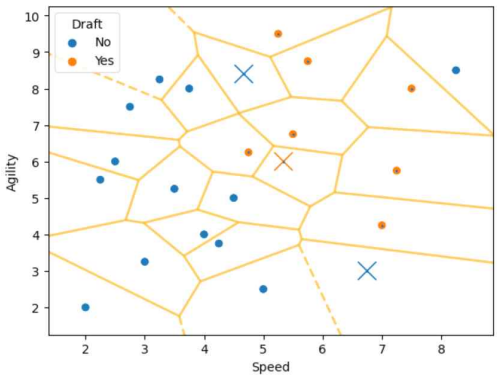


3-NN Manhattan distance

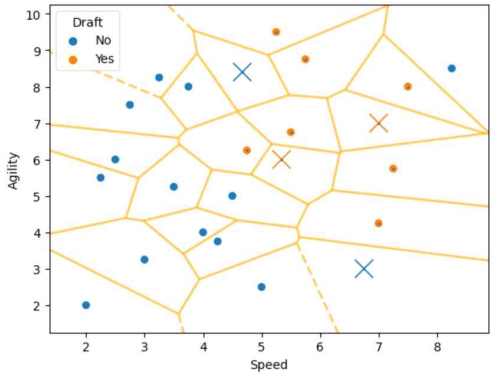


3-NN Euclidean distance

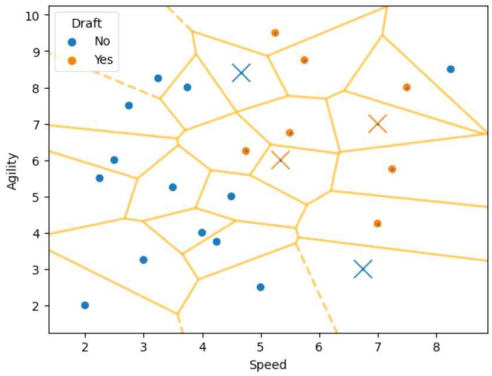
앞서 봤던 것처럼 Manhattan distance를 사용한 버전에는 [4.67, 8.4]의 'Draft' 값이 'No'이고, Euclidean distance를 사용한 버전에는 'Yes'임을 볼 수 있다.



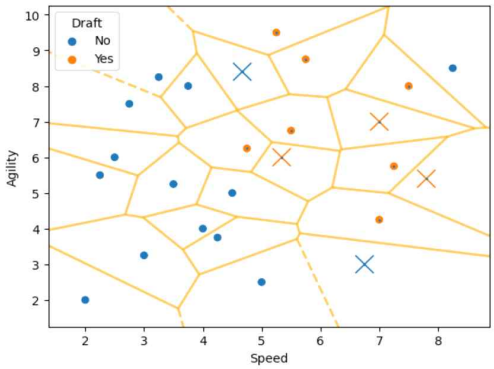
데이터가 3개 추가된 그래프



[7.0, 7.0]값이 추가된 그래프



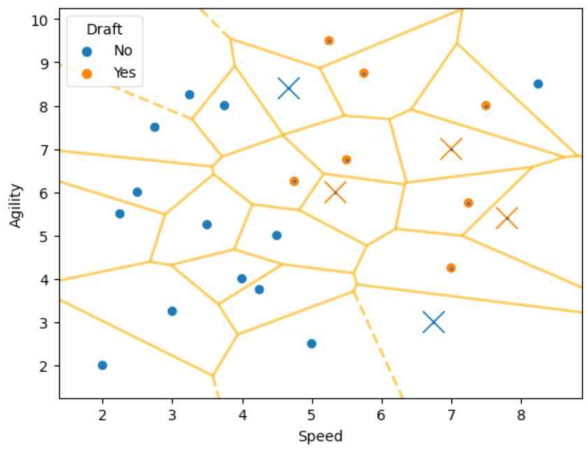
데이터 4개가 추가된 그래프



[7.8, 5.4]값이 추가된 그래프

따라서, 5개의 데이터를 모두 포함한 3-NN Manhattan 버전의 그래프는 다음과 같다.

```
fig = voronoi_plot_2d(vor, show_vertices=False, line_colors='orange', line_width=2, line_alpha=0.6, point_size=2)
fig = sns.scatterplot(data = c, x='Speed', y='Agility', hue='Draft', s=50)
fig = sns.scatterplot(data = d, x='Speed', y='Agility', hue='Draft', marker='x', s=220, legend=False)
plt.show()
```



4. 5-NN Manhattan distance

4-1) 3-NN Manhattan distance와 다른점

코드 부분에서는 큰 차이가 없다. 결과에서 차이점만 보자.

```
[7.0, 4.25, 'Yes']
[5.0, 2.5, 'No']
[4.25, 3.75, 'No']
[7.25, 5.75, 'Yes']
[4.0, 4.0, 'No']
Expected: No

[4.75, 6.25, 'Yes']
[5.5, 6.75, 'Yes']
[4.5, 5.0, 'No']
[7.25, 5.75, 'Yes']
[3.5, 5.25, 'No']
Expected: Yes

[3.75, 8.0, 'No']
[5.75, 8.75, 'Yes']
[3.25, 8.25, 'No']
[5.25, 9.5, 'Yes']
[4.75, 6.25, 'Yes']
Expected: Yes

[7.5, 8.0, 'Yes']
[7.25, 5.75, 'Yes']
[5.5, 6.75, 'Yes']
[5.34, 6.0, 'Yes']
[8.25, 8.5, 'No']
Expected: Yes

[7.25, 5.75, 'Yes']
[7.0, 4.25, 'Yes']
[7.0, 7.0, 'Yes']
[7.5, 8.0, 'Yes']
[5.34, 6.0, 'Yes']
Expected: Yes
```

<5-NN Manhattan>

결측값.
영향 X

```
[7.0, 4.25, 'Yes']
[5.0, 2.5, 'No']
[4.25, 3.75, 'No']
Expected: No

[4.75, 6.25, 'Yes']
[5.5, 6.75, 'Yes']
[4.5, 5.0, 'No']
Expected: Yes

[3.75, 8.0, 'No']
[5.75, 8.75, 'Yes']
[3.25, 8.25, 'No']
Expected: No

[7.5, 8.0, 'Yes']
[7.25, 5.75, 'Yes']
[5.5, 6.75, 'Yes']
Expected: Yes

[7.25, 5.75, 'Yes']
[7.0, 4.25, 'Yes']
[7.0, 7.0, 'Yes']
Expected: Yes
```

<3-NN Manhattan>

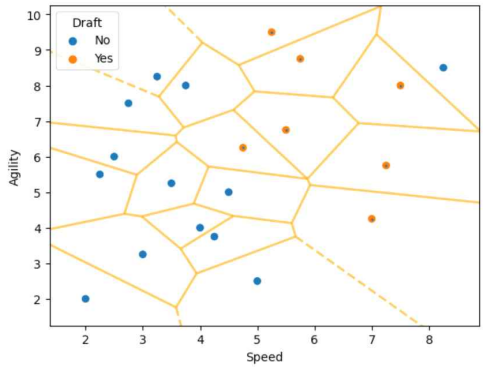
k가 3인 경우에는 'No'였지만,
k가 5인 경우, [4.67, 8.4]의 최근접 값들은 'Draft'가 'Yes'인 값들이 많기 때문에 [4.67, 8.4]의 'Draft'값은 'Yes'가 된다.

따라서, k가 5인 경우와 3인 경우에서 서로 다름을 알 수 있다.

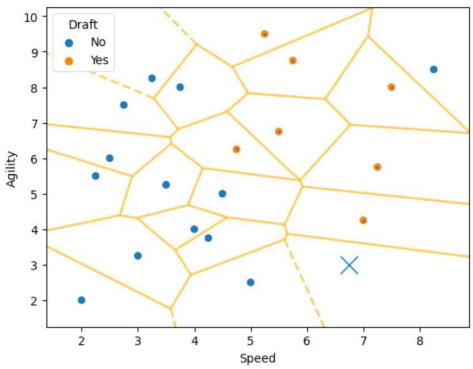
4-2) Voronoi 그래프 그리기

df를 loc함수를 사용하여 원하는 행들을 추출해서 그래프를 그려본다.

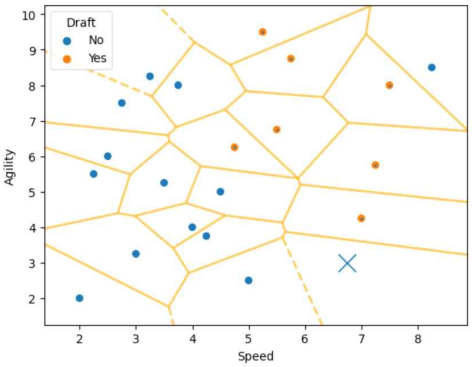
새로운 값이 추가될 때마다 어떻게 그래프가 변하는지 비교해보자.



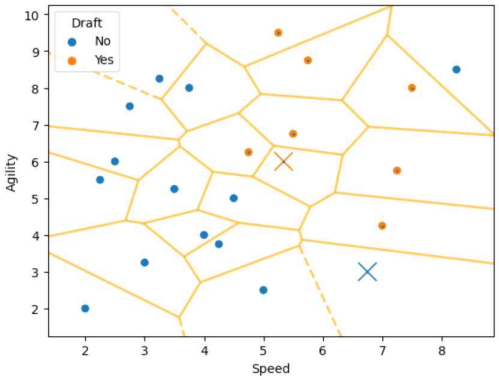
처음 20개의 데이터로 그린 그래프



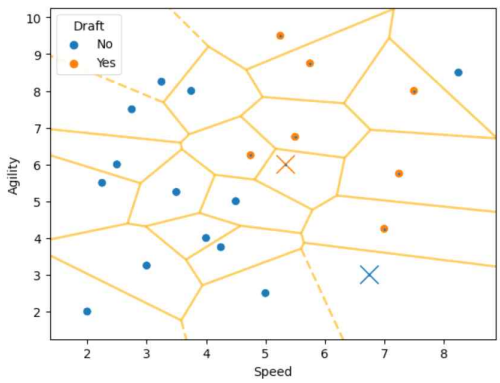
[6.75,3]값이 추가된 그래프



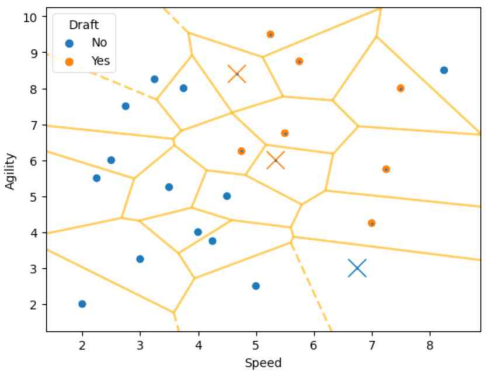
데이터가 1개 추가된 그래프



[5.34, 6.0]값이 추가된 그래프



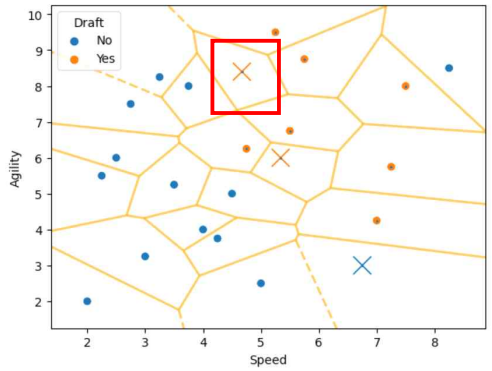
데이터가 2개 추가된 그래프



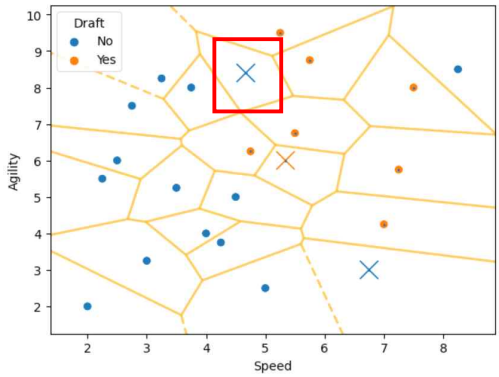
[4.67, 8.4]값이 추가된 그래프

세 번째로 추가되는 [4.67, 8.4]에서 3-NN Manhattan distance와 차이점이 나타난다.

다음은 [4.67, 8.4]값이 추가 될 때 5-NN Manhattan distance와 3-NN Manhattan distance의 voronoi그래프이다.

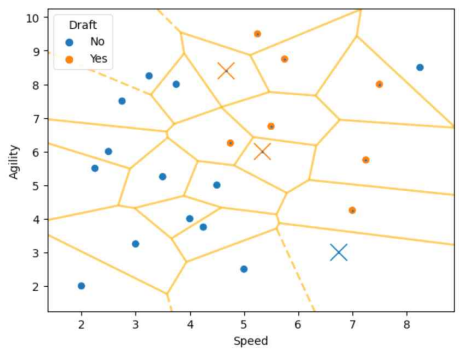


5-NN Manhattan distance

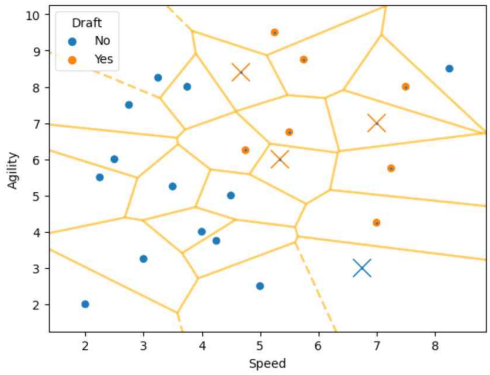


3-NN Manhattan distance

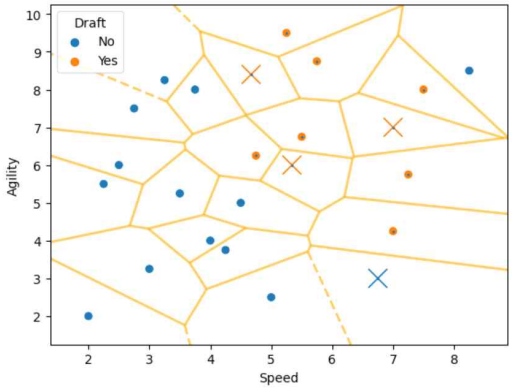
앞서 봤던 것처럼 5-NN Manhattan distance를 사용한 버전에는 [4.67, 8.4]의 'Draft' 값이 'Yes'이고, 3-NN Manhattan distance를 사용한 버전에는 'No'임을 볼 수 있다.



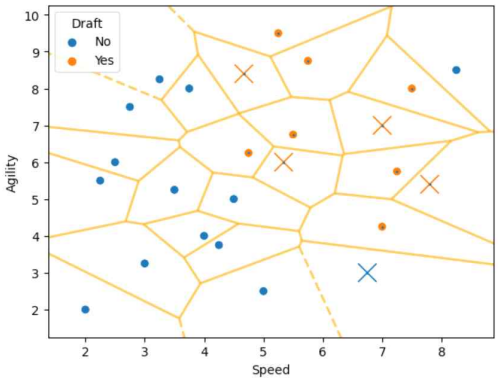
데이터가 3개 추가된 그래프



[7.0, 7.0]값이 추가된 그래프



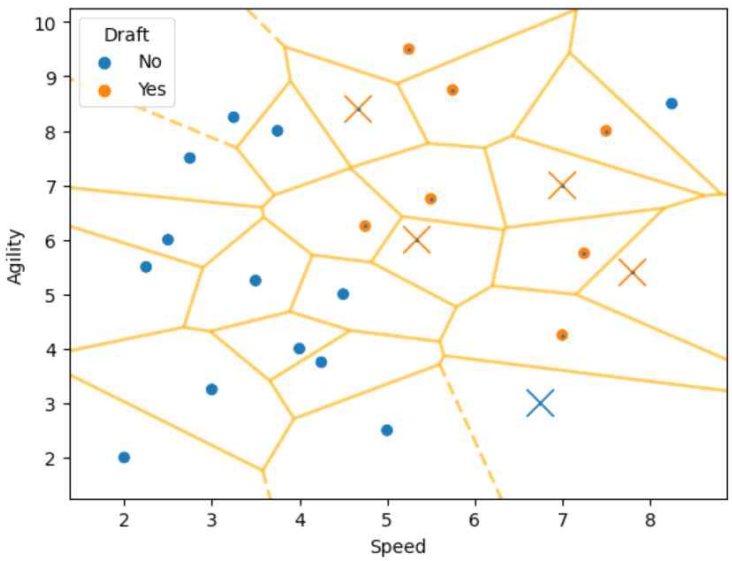
데이터가 4개 추가된 그래프



[7.8, 5.4]값이 추가된 그래프

따라서, 5개의 데이터를 모두 포함한 5-NN Manhattan 버전의 그래프는 다음과 같다.

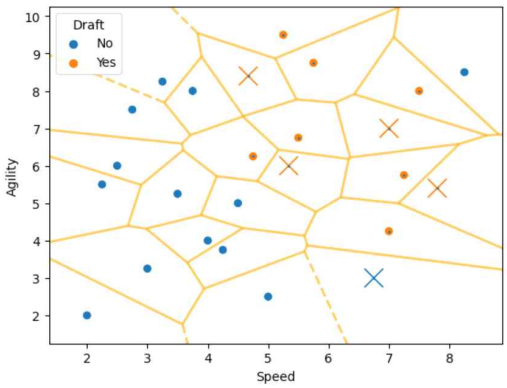
```
fig = voronoi_plot_2d(vor, show_vertices=False, line_colors='orange', line_width=2, line_alpha=0.6, point_size=2)
fig = sns.scatterplot(data = c, x='Speed', y='Agility', hue='Draft', s=50)
fig = sns.scatterplot(data = d, x='Speed', y='Agility', hue='Draft', marker='x', s=220, legend=False)
plt.show()
```



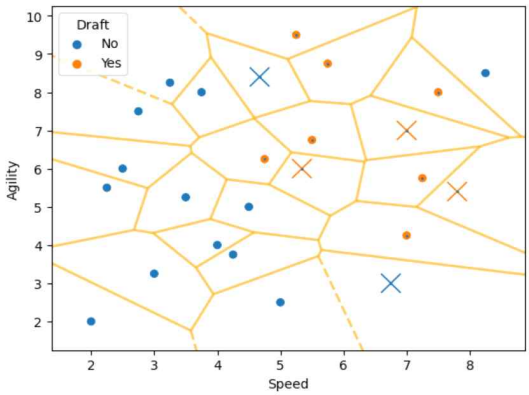
결과

4개 버전 Voronoi 그래프의 유사점과 차이점

1) 3-NN Euclidean과 3-NN Manhattan의 유사점과 차이점



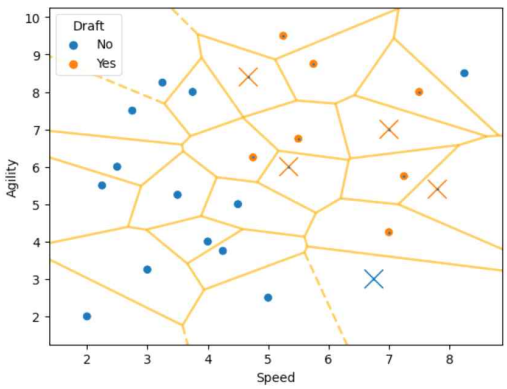
3-NN Euclidean distance



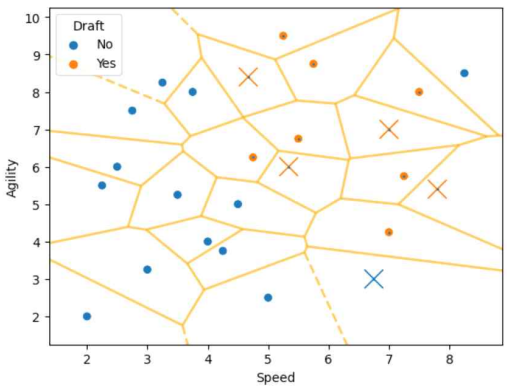
3-NN Manhattan distance

앞서 보았듯이 세 번째로 추가되는 [4.67, 8.4]에서 차이점이 나타났었다.
[4.67, 8.4]값의 최근접 값이 달랐기 때문에 그에 따라 'Draft'값이 달랐었다.
나머지 값들([6.75,3], [5.34, 6.0], [7.0,7.0], [7.8, 5.4])의 'Draft'값은 모두 동일하기에 [4.67, 8.4]을 제외하곤 그래프가 같은 모습을 볼 수 있다.

2) 3-NN Euclidean과 5-NN Euclidean의 유사점과 차이점



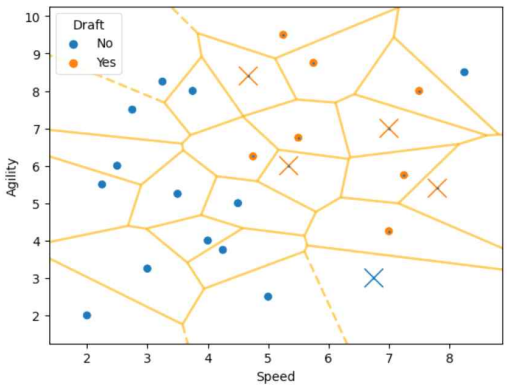
3-NN Euclidean distance



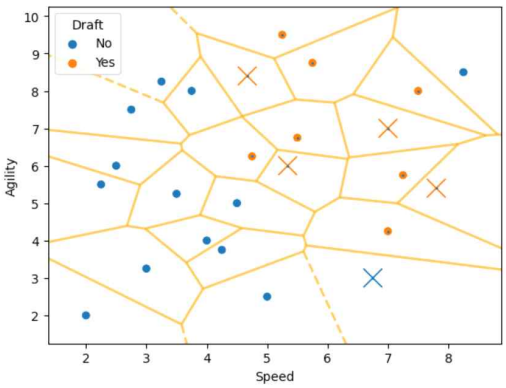
5-NN Euclidean distance

두 버전은 추가되는 값들의 'Draft'값이 모두 동일했었다.
따라서 Voronoi 그래프 모양도 동일함을 볼 수 있다.

3) 3-NN Euclidean과 5-NN Manhattan의 유사점과 차이점



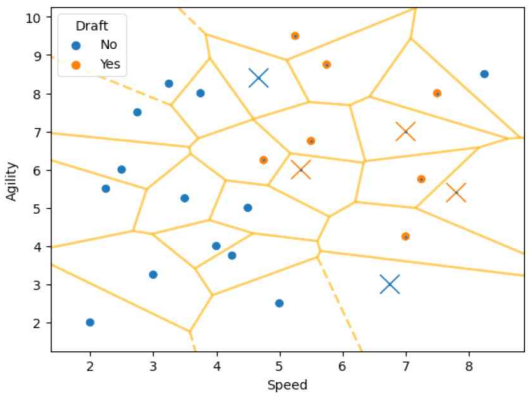
3-NN Euclidean distance



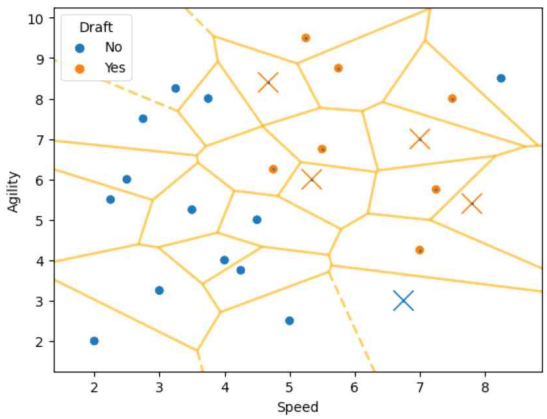
5-NN Manhattan distance

두 버전은 추가되는 값들의 'Draft'값이 모두 동일했었다.
따라서 Voronoi 그래프 모양도 동일함을 볼 수 있다.

4) 3-NN Manhattan과 5-NN Euclidean의 유사점과 차이점



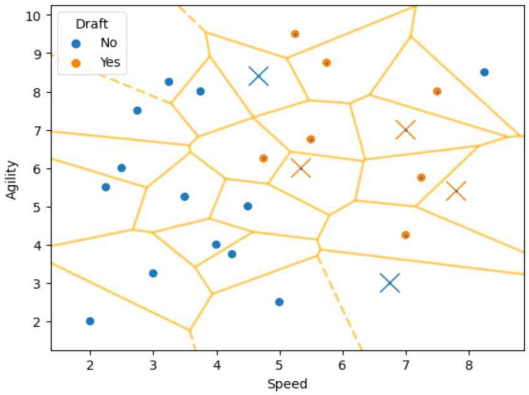
3-NN Manhattan distance



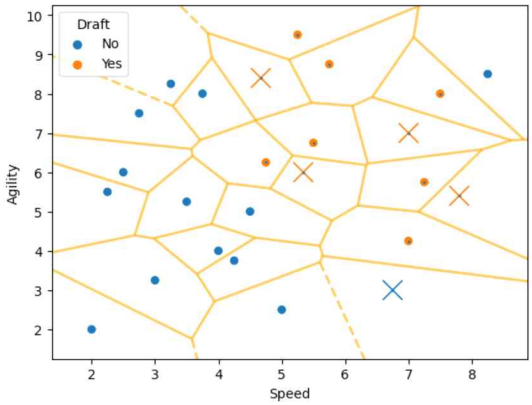
5-NN Euclidean distance

세 번째로 추가되는 [4.67, 8.4]에서 차이점이 나타났었다.
[4.67, 8.4]값의 최근접 값이 달랐기 때문에 그에 따라 'Draft'값이 달랐었다.
나머지 값들([6.75,3], [5.34, 6.0], [7.0,7.0], [7.8, 5.4])의 'Draft'값은 모두 동일하기에 [4.67, 8.4]을 제외하곤 그래프가 같은 모습을 볼 수 있다.

5) 3-NN Manhattan과 5-NN Manhattan의 유사점과 차이점



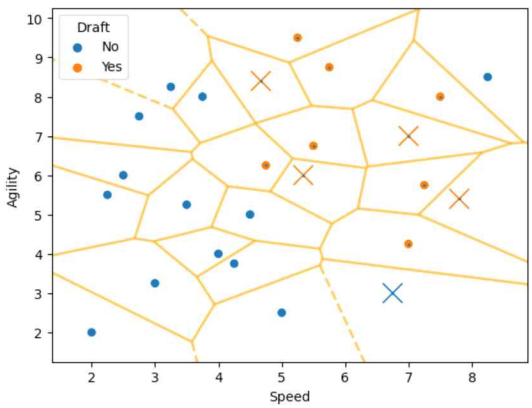
3-NN Manhattan distance



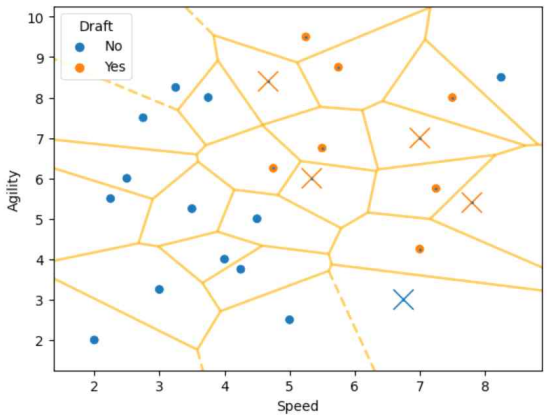
5-NN Manhattan distance

마찬가지로, 세 번째로 추가되는 [4.67, 8.4]에서 차이점이 나타났었다.
나머지 값들([6.75,3], [5.34, 6.0], [7.0,7.0], [7.8, 5.4])의 'Draft'값은 모두 동일하기에 [4.67, 8.4]을 제외하곤 그래프가 같은 모습을 볼 수 있다.

6) 5-NN Euclidean과 5-NN Manhattan의 유사점과 차이점



5-NN Manhattan distance



5-NN Euclidean distance

두 버전은 추가되는 값들의 'Draft'값이 모두 동일했었다.
따라서 Voronoi 그래프 모양도 동일함을 볼 수 있다.

고찰 및 결론

Euclidean distance 함수와 Manhattan distance함수를 구현했다. 3-NN 과 5-NN 알고리즘을 구현하는데 거리를 계산하는 방법으로 Euclidean distance 함수와 Manhattan distance 함수를 사용했다. 그 결과 4개 버전을 얻을 수 있었다.

각 버전별로 새로 추가되는 데이터들에 대해 'Draft'값을 판별하는 지 확인했다.

처음에는 4개 버전에서 판별하는 'Draft'값들이 대부분 비슷할 것이라고 예상했다. 이유는 거리 계산 함수가 달라도 최근접 값은 거의 동일하게 나올 것이라 생각했고, 'Draft'값 또한 같게 나올 것이라 예상했었다. 하지만 예상은 반은 맞고 반은 틀렸다. 추가되는 값들의 최근접 값들을 출력해본 결과 대부분 동일 할거란 예상은 맞았다. 하지만 몇 개의 다른 최근접 값들의 'Draft' 값의 영향을 받아서 추가되는 값의 'Draft'가 서로 다른 모습을 확인할 수 있었다.

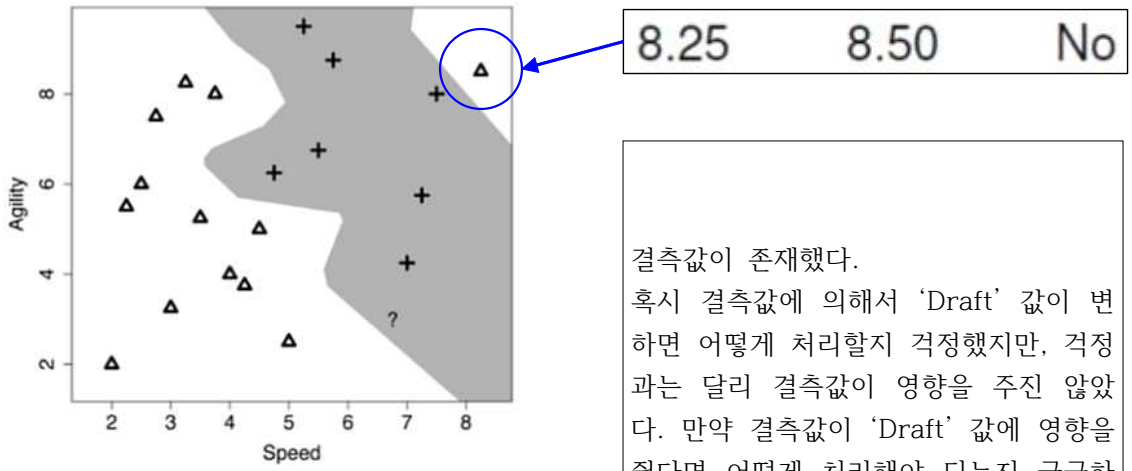
처음에 dataset과 testset을 설정할 때 리스트로 만들어줬었다. 하지만 Voronoi 그래프를 그리면서 생각이 들었다. 어차피 그래프를 그리려면 데이터 프레임으로 넣어줘야 하기에 애초에 리스트가 아닌 데이터 프레임으로 만들어서 구현했으면 좀 더 편하게 코딩할 수 있지 않았을까 생각이 든다.

추가되는 데이터에 따라 그래프 모양이 달라지는 것을 확인하기 위해 데이터가 기존 dataset에 추가되는 시점에서 그래프를 그리게해서 5개를 한번에 그리도록 구현한 것이 아닌, loc함수를 사용하여 원하는 행을 추출해서 그래프를 그렸다. 이 부분에서 코딩 실력이 부족함을 느꼈다.

최근접 값들의 'Draft' 값들의 수가 아닌 각 근접 값들의 순위에 따른 다른 가중치를 부여하는 것이 필요하다고 느꼈다.
3-NN Manhattan과 3-NN Euclidean을 비교할 때 제일 마지막 근접값(3등으로 가까운 값)의 'Draft'값이 달라서 추가되는 값의 'Draft'가 서로 달랐는데 만약 가중치를 매긴다면 어떻게 'Draft'를 판별했을지 궁금하다. 살짝 생각해

보자면 가중치의 편차도 중요할 것 같다.
그리고 3-NN Manhattan과 5-NN Manhattan에서 세 번째로 추가되는 [4.67, 8.4]에서 차이점이 나타났었다. 이유는 3-NN Manhattan에서는 최근접 값들의 'Draft'값이 ['No', 'Yes', 'No']여서 'No'로 판별했지만, 5-NN Manhattan에서는 ['No', 'Yes', 'No', 'Yes', 'Yes']여서 'Yes'로 판별했다.
만약, 가중치를 매겼더라면 어떻게 'Draft'를 판별했을지 궁금하다.
내가 임의로 가중치를 매겨보겠다. 1등은 10점, 2등은 8점순으로, 5등은 2점으로 하겠다. 그러면 5-NN Manhattan은 [4.67, 8.4]의 'No' 가중치는 1등+3등 = 16점, 'Yes' 가중치는 2등+4등+5등 = 8+4+2 = 14점으로 'No'가 되는 것을 볼 수 있다. 즉, 결과가 다르게 나타나는 것을 볼 수 있다. 추후에 가중치를 포함하는 k-NN 알고리즘을 구현해보고 싶다.

또는, 유클리드와 맨허튼 거리를 합친 거리가 있으면 좋겠다 생각을 했다. 구글링을 해봤는데 이미 존재했다. Minkowski distance라는 유클리드와 맨허튼 거리를 일반화한 거리 계산법이 존재했다. Minkowski distance로 계산한 3-NN과 5-NN의 Voronoi 그래프를 확인하고 Euclidean과 Manhattan과 비교해서 차이점을 확인하고 싶었지만 시험 기간이기에 확인은 추후로 미뤄본다.



결측값이 존재했다.
혹시 결측값에 의해서 'Draft' 값이 변하면 어떻게 처리할지 걱정했지만, 걱정과는 달리 결측값이 영향을 주진 않았다. 만약 결측값이 'Draft' 값에 영향을 줬다면 어떻게 처리해야 되는지 궁금하다. 만약 영향을 줬다면 나였으면 [8.25, 8.50, 'No'] 데이터를 삭제했을 것이다.