1 k-최근접 이웃

k-최근접 이웃(KNN) 알고리즘

1. 예측변수들이 유사한 k 개의 레코드를 찾는다.
2. 분류 : 이 유사한 레코드들 중에 다수가 속한 클래스가 무엇인지 찾은 후에 새로운 레코드를 그 클래스에 할당한다.
3. 예측 : 유사한 레코드들의 평균을 찾아서 새로운 레코드에 대한 예측값으로 사용한다.

이웃 : 예측변수에서 값들이 유사한 레코드
거리 지표 : 각 레코드 사이가 얼마나 멀리 떨어져 있는지를 나타내는 단일 값

KNN은 가장 간단한 예측 분류 방법중 하나. 회귀와 달리 피팅하는 과정이 필요 없다. 특징이 어떤 척도에 존재하는지 가까운 정도를 어떻게 측정할 것인지 kfmf 어떻게 설정한 것인지에 따라 결과가 나뉜다. 모든 예측변수들은 수치형이어야 한다.

2 거리 지표

유사성은 거리지표를 통해 결정된다. 유클리드 거리가 가장 많이 사용되고 다음으로 많이 사용되는 거리 지표는 맨해튼 거리이다.
다른 거리 지표
마할라노비스 거리 두변수간의 상관관계를 사용하기 때문에 장점이 있다. 유클리드 거리와 맨해튼 거리는 상관성을 고려하지 않아 상관성이 있는 변수에서 원인이 되는 속성에 더 큰 가중치를 두게 된다. 마할라노비스 거리는 주성분 사이의 유클리드 거리를 의미한다.
단점 : 복잡성 증가 및 많은 계싼 필요. 계산에 공분산행렬을 사용.

3 원핫 인코더

예측변수가 수치형이어야 하기 때문에 원핫 인코더를 이용해서 범주형 데이터를 수치형 데이터로 변환한다.
선형 회귀나 로지스틱 회귀에서 원핫 인코딩은 다중공신성 관련된 문제를 일으킨다. 이런 경ㅇ우 가변수를 생략한다.

4 표준화

표준화에는 MinMaxScaler와 StandardScaler

5 k 선택하기

k가 너무 작으면 데이터의 노이즈 성분까지 고려하는 오버피팅 문제가 발생
k가 너무 크면 결정함수가 너무 과하게 평탄화되어 데이터의 지역 정보를 예측하는 KNN 기능을 잃어버리게 된다.
최적의 k를 찾기 위해 정확도 지표들을 활용. 특히 홀드아웃 데이터 또는 타당성 검사를 위해 따로 떼어놓은 데이터에서의 정확도를 가지고 k 값을 결정하는 데 사용한다. 동률이 나오는 경우 홀수를 사용.

6 KNN을 통한 피처 엔지니어링

지역적 정보를 추가하기 위해 KNN 사용.
KNN은 데이터에 기반하여 분류 결과를 얻는다.
이 결과는 해당 레코드에 새로운 특징으로 추가된다. 이 결과를 다른 분류방법에 사용한다. 원래의 예측변수들을 두번씩 사용하는 셈이다.

KNN

July 19, 2022

[18]:
```
import pandas as pd
from sklearn.datasets import load_breast_cancer
breast_cancer_data = load_breast_cancer()
df_data = pd.DataFrame(breast_cancer_data.data)
df_labels = pd.DataFrame(breast_cancer_data.target)
```

[19]:
```
df_data.head()
```

[19]:

	0	1	2	3	4	5	6	7	8	\
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	

	9	…	20	21	22	23	24	25	26	27	\
0	0.07871	…	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	
1	0.05667	…	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	
2	0.05999	…	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	
3	0.09744	…	14.91	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	
4	0.05883	…	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	

	28	29
0	0.4601	0.11890
1	0.2750	0.08902
2	0.3613	0.08758
3	0.6638	0.17300
4	0.2364	0.07678

[5 rows x 30 columns]

[17]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
```

[21]:
```
x_train,x_test,y_train,y_test =␣
↪train_test_split(df_data,df_labels,random_state=42,shuffle=True)
```

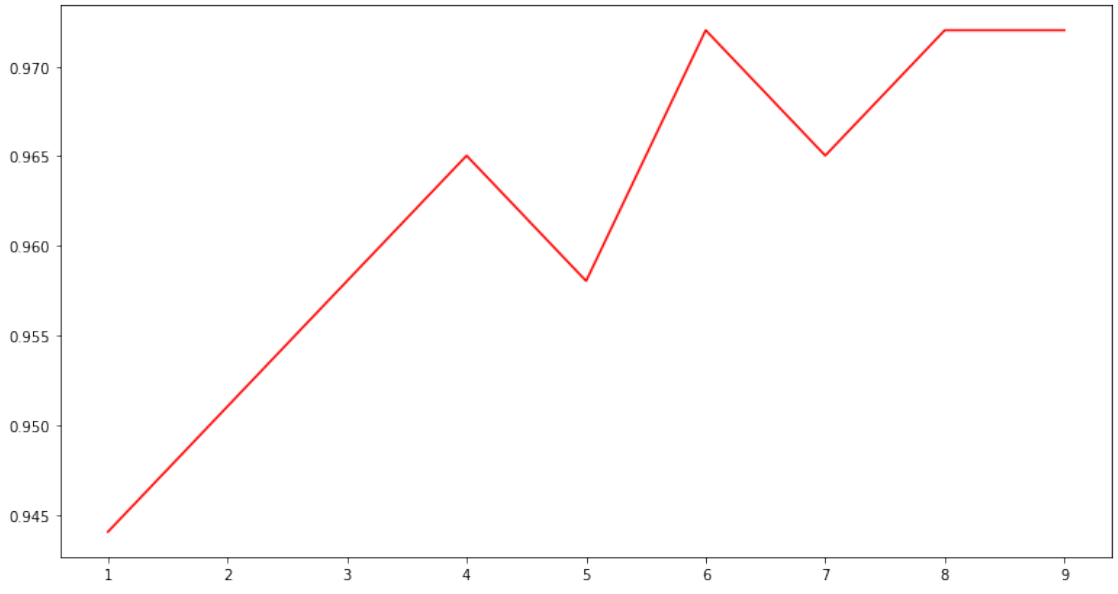
[24]:
```
y_train
```

[24]:
```
     0
287  1
512  0
402  1
446  0
210  0
..  ..
71   1
106  1
270  1
435  0
102  1

[426 rows x 1 columns]
```

[22]:
```
scaler = StandardScaler()
x_train_s = scaler.fit_transform(x_train)
x_test_s=scaler.fit_transform(x_test)
```

[53]:
```
acc = []
for k in range(1,10):
    knn=KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train_s,y_train.values.ravel())
    acc.append(knn.score(x_test_s,y_test))
```

[47]:
```
import matplotlib.pyplot as plt

ax, fig = plt.subplots(1,1,figsize = (13,7))
plt.plot([i for i in range(1,10)],acc,color = 'red')
plt.xticks([i for i in range(1,10)])
plt.show()
```



[51]:
```
print(acc[5])
```

```
0.972027972027972
```