심혈관 질환 발병 확인 모델

바이오메디컬공학부 201800308 강민구, 201803952 황교준

1. 심혈관 질환 발병 확인 모델의 동기

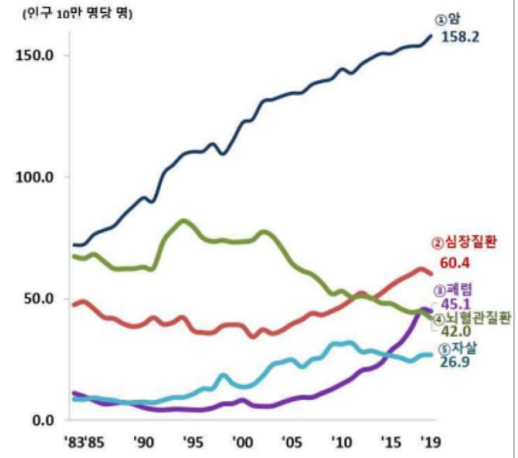
사회가 점점 발달함에 따라 우리의 삶이 윤택해지고, 수명이 늘어났다. 하지만 사회의 발달에도 안 좋은 습관(담배, 술 등)으로 아직 우리는 많은 위험에 노출되어 있는데 큰 위험 중 하나가 심혈관 질환이다.

아무 이상징후 없이 갑자기 찾아오는 경우가 많은 심혈관 질환은 2019년 통계청의 사망 원인 통계 조사의 결과에서 암 다음으로 높은 사망률을 가지는 원인이었으며, 10만 명당 60.4명의 사망자를 만들 정도로 위험한 질병이다. 조용하지만 몹시 치명적이기에 원인을 알고 예방하는 것이 어떤 질병보다 중요하다. 심혈관 질병을 예방하기 가장 좋은 방법은 직접 병원에 가 검사를 해보는 것이지만, 바쁜 사람이나 해외에 있는 사람들은 검사를 받기 힘들 수 있다. 따라서 집에서 간단하게 테스트를 해보고 자신이 심혈관 질병이 있는지 알려주는 모델을 만들면 병원에 가야하는 불편함을 줄여줄 수 있을 것이다.

<사망원인 순위 추이>

(단위: 인구 10만 명당 명)

순위	사망원인	사망률	'18년 순위 대비
1	악성신생물(암)	158.2	-
2	심장 질환	60.4	-
3	폐렴	45.1	-
4	뇌혈관 질환	42.0	-
5	고의적 자해(자살)	26.9	-
6	당뇨병	15.8	-
7	알츠하이머병	13.1	⬆(+2)
8	간 질환	12.7	⬇(-1)
9	만성 하기도 질환	12.0	⬇(-1)
10	고혈압성 질환	11.0	-



<사망원인 통계 조사, 통계청, 2019>

우리가 원하는 심혈관 질환 발병 모델은 사용자가 자신의 data를 입력하면 다른 환자의 data들로부터 학습된 심혈관 질환 모델이 사용자의 심혈관 질병의 발병 여부에 대해 결과로 알려주는 것이다. 이러한 심혈관 질환 발병 확인 모델을 만들기 위해 첫번째로 noisy data나, missing data를 찾아서 수정을 하는 데이터 전처리를 하고, 두번째로 attribute들의 상관관계를 조사해서 attribute들을 제거해 보고, 세번째로 수정된 data를 가지고 decision tree 와 random forest classification으로 model의 성능을 알아볼 것이다.

2. 데이터 전처리

사용한 데이터는 Kaggle에서 가져온 심장질환 발병 여부 data이고 id변수를 포함한 총 13개의 attribute를 가지고 있다. 또한 7만명의 환자에 대한 7만개의 object가 있다. 이때의 attribute는 아래와 같다.

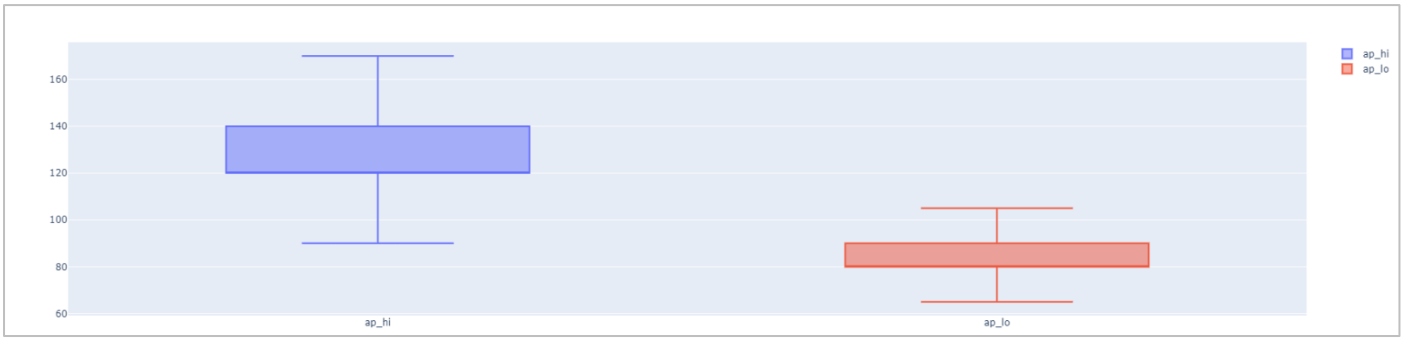
No.	Feature	No.	Feature
1	age(DAY)	7	Cholesterol (1 : normal, 2 : above normal, 3 : well above normal)
2	gender(1-women, 2-men)	8	gluc(혈당) (1 : normal, 2 : above normal, 3 : well above normal)
3	Height(cm)	9	smoke(0-비흡연, 1-흡연)
4	Weight(Kg)	10	alco(0-음주X, 1-음주O)
5	ap_hi (수축혈압)	11	active(운동여부) (0-운동X, 1-운동O)
6	ap_lo (이완혈압)	12	Cardio(target) (0-발병X, 1-발병O)

이때 id란 변수는 모델 학습에 필요 없는 변수이므로 제거했다. 또한 age attribute를 보면 살아온 날짜로 되어 있다. 따라서 입력하는 사람의 편의성을 위해 원래 age변수를 365로 나누어 주고 소수점 첫째자리에서 반올림해서 나이로 변수들을 바꿔주었다. 또한 모델 학습을 위하여 일관성 있는 수치를 선택하였는데 키와 몸무게를 선택하기 보다는 bmi라는 일관성 있는 수치를 선택하는 것이 좋을 것이라 생각했다. cm단위로 되어있는 height를 100으로 나누어 준 값의 제곱으로 weight를 나눠서 bmi지수를 얻었다. bmi지수를 얻은 후, drop함수를 통하여 변수 height, weight를 제거했다.

또한 일반적으로 수축 혈압(ap_hi)은 이완 혈압(ap_lo)보다 높기 때문에 data에서 이완 혈압이 더 높은 경우를 noisy data로 판단하고 제거하였다. 또한 혈압 attribute(ap_hi, ap_lo)의 데이터를 보니(클릭) 최솟값이 음수이고, 최대값이 지나치게 큰 혈압을 나타내는 outlier을 가지고 있어 데이터의 이상치와 불가능한 값들을 없애 줄 필요가 있다. 이를 위해 boxplot을 그려보았으며 그때의 결과는 <그림1>과 같았다.

<그림 1>

boxplot에서 각 attribute의 upper fence와 lower fence 범위에 해당하지 않는 outlier들을 삭제하기 위해서 수축기 혈압(ap_lo)에서 90보다 작거나 170보다 큰 이상치들을 제거해 주고, 이완기 혈압(ap_lo)이 65보다 작거나 185보다 큰 경우를 제거해주었다. 전처리 후의 혈압 attribute boxplot은 <그림2>와 같다.



<그림 2>

그 결과 64,500개의 object가 남았으며, 다른 attribute에 대해서는 각각의 변수들의 개수의 총합이 64,500 임을 확인하여 missing value가 없음을 알 수 있었다. 이때 target의 분포는 cardio(0) : cardio(1) = 32354 : 32146으로 balanced 했다.

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
69995	99993	19240	2	168	76.0	120	80	1	1	1	0	1	0
69996	99995	22601	1	158	126.0	140	90	2	2	0	0	1	1
69997	99996	19066	2	183	105.0	180	90	3	1	0	1	0	1
69998	99998	22431	1	163	72.0	135	80	1	2	0	0	0	1
69999	99999	20540	1	170	72.0	120	80	2	1	0	0	1	0

70000 rows × 13 columns

<원본 데이터>

	age	gender	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio	BMI
0	50	2	110	80	1	1	0	0	1	0	21.97
1	55	1	140	90	3	1	0	0	1	1	34.93
2	52	1	130	70	3	1	0	0	0	1	23.51
3	48	2	150	100	1	1	0	0	1	1	28.71
5	60	1	120	80	2	2	0	0	0	0	29.38
...	...	...	...	...	...	...	...	...	...	...	...
69994	58	1	150	80	1	1	0	0	1	1	29.38
69995	53	2	120	80	1	1	1	0	1	0	26.93
69996	62	1	140	90	2	2	0	0	1	1	50.47
69998	61	1	135	80	1	2	0	0	0	1	27.10
69999	56	1	120	80	2	1	0	0	1	0	24.91

64500 rows × 11 columns

<전처리 된 데이터>

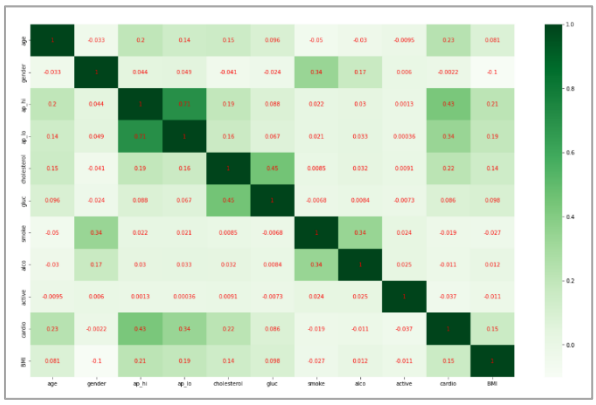
3. 연관관계 분석

전처리한 데이터를 사용하여 연관관계 분석을 진행하였다. 상관관계 분석, support/confidence/lift 분석, kulczynski /Imbalanced Ratio/chi-square 분석과 PCA 분석을 통해 classification 과정에서 사용할 attribute를 추출했다.

.

3.1 상관관계(Correlation) 분석

상관관계는 그 값이 0보다 큰가 작은가를 통해 두 attribute의 positive 혹은 negative 관계 정도를 알아보는 데 사용한다. 파이썬의 .corr( ) 함수를 사용하였고, 분석을 위해 seaborn library를 통해 시각화 했다. 결과는 <그림 3>과 같다. 색이 진할수록 두 attribute 사이의 상관관계가 높음을 의미한다. 예측하려는 target이 cardio 이기 때문에, 다른 attribute와 cardio 간의 관계를 살펴보았다. 그 결과 상관계수가 양수인 attribute와 음수인 attribute를 구할 수 있었으며, 상관계수의 절댓값이 0.2 이상으로 어느 정도의 상관관계가 있으리라 예상되는 attribute 역시 추출하였다. 이는 <표 1>에서 확인할 수 있다.



<그림 3>

기준	attribute
Corr > 0	age, ap_hi, ap_lo, cholesterol, gluc, BMI
Corr < 0	gender, smoke, alce, active
|Corr| > 0.2	age, ap_hi, ap_lo, cholesterol

<표 1>

또한 <그림 3>에서 알아낼 수 있는 것이 있다. 보통 혈압과 콜레스테롤, 혈당이 서로 관련이 있을 거라고 생각한다. 하지만 이 데이터에서는 gluc과 cholesterol은 서로 0.45로 높은 상관관계를 가지지만 혈압(ap_hi, ap_lo)과는 큰 관계가 없음을 확인할 수 있다. 물론 임의의 표본에 대한 데이터이기에 일반화는 어렵지만, 통념을 부정하는 결과가 나온 것에 의미가 있다고 생각한다. <표 1>을 바탕으로 추출한 attribute는 <결과 1>과 같다.

<결과 1> : [age, ap_hi, ap_lo, cholesterol]

3.2 support / confidence / lift 분석

confidence는 A를 포함하는 거래내역 중 B가 포함된 비율(P(B|A))로써 두 집합이 동시에 나온다는 규칙의 신뢰도에 대한 척도이다. 두번째로 support는 A와 B를 동시에 포함하는 비율로써 confidence를 지지하는 척도이다. 즉, confidence에 의한 규칙이 지지받기 위해서는 support값이 높아야 한다. 마지막으로 lift는 confidence 규칙이 의미 있는지 확인하기 위한 척도이다. 만약 confidence 값이 P(B)와 같으면 A와 B가 서로 독립이 되기 때문에 A와 B는 아무 관계가 없다. 따라서 confidence를 P(B)로 나눠 lift를 구하고, 규칙이 의미가 있는지 판단한다. Lift가 1이면 두 attribute는 서로 독립이고, 1보다 크면 서로 양의 관계를, 1보다 작으면 서로 음의 관계를 가진다.

후술할 연산을 위해 cardio 데이터를 범주형 데이터와 트랜잭션 데이터로 바꿔준다. cardio데이터의 각 attribute가 가지는 모든 경우를 고려하여 binary 혹은 float 자료형의 데이터를 각각 범주형 자료로 바꾼다. 예를 들어 BMI 수치는 분류 기준에 따라 저체중, 정상, 과체중, 비만, 고도비만의 범주로 나눠 처리해준다. Age는 10으로 나눠 연령대를 적용함으로써 범주형 데이터로 변경해주었다. 그 다음은 mlxtend 메소드의 transform 함수를 사용하여 범주형 데이터를 트랜잭션 데이터로 바꿔준다. 그 결과 범주형 데이터의 각 attribute가 가지는 category들이 트랜잭션의 attribute로 들어가고, 해당 값이 True 또는 False 가 되는 것을 <그림 4>에서 확인할 수 있다. 또한 각각의 category들이 총 29개의 case로 분류된다.

	30	40	50	60	Above_Normal_cho	Above_Normal_gluc	Active	Alcohol	Cardio	HBP_DIAS	...	No_Smoke	No_cardio	Normal_cho	Normal_gluc	OBESITY	OVER	Smoke	Well_Above_Normal_ch
0	False	False	True	False	False	False	True	False	False	False	...	True	True	True	True	False	False	False	False
1	False	False	True	False	False	False	True	False	True	True	...	True	False	False	True	True	False	False	True
2	False	False	True	False	False	False	False	False	True	False	...	True	False	False	True	False	False	False	True
3	False	True	False	False	False	False	True	False	True	True	...	True	False	True	True	False	True	False	False
4	False	False	False	True	True	True	False	False	False	False	...	True	True	False	False	False	True	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
64495	False	False	True	False	False	False	True	False	True	False	...	True	False	True	True	False	True	False	False
64496	False	False	True	False	False	False	True	False	False	False	...	False	True	True	True	False	True	True	False
64497	False	False	False	True	True	True	True	False	True	True	...	True	False	False	False	False	False	False	False
64498	False	False	False	True	False	True	False	False	True	False	...	True	False	True	False	False	True	False	False
64499	False	False	True	False	True	False	True	False	False	False	...	True	True	False	True	False	False	False	False

64500 rows × 29 columns

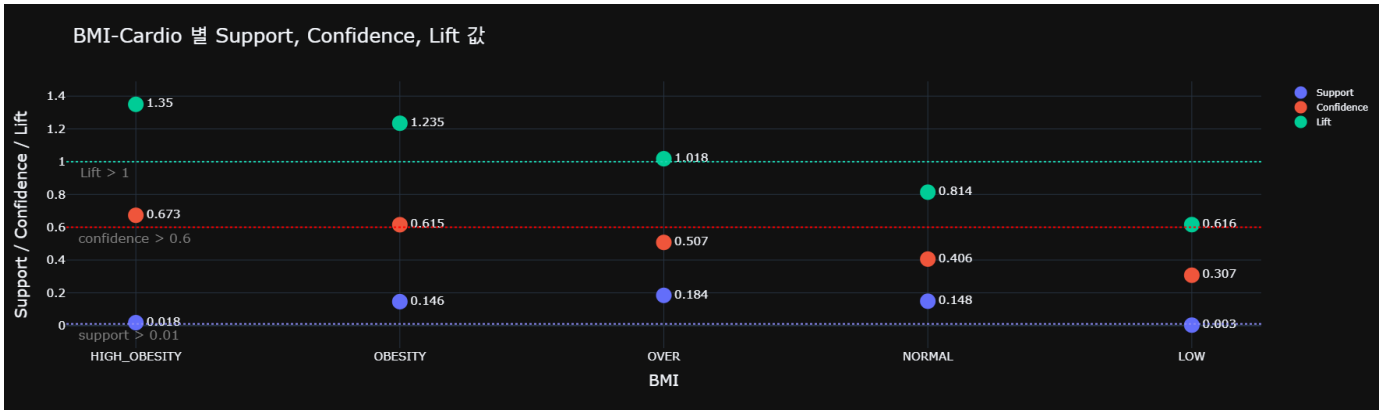
<그림 4>

이제 mlxtend.frequent_patterns 모듈의 apriori, association_rules 함수를 사용해 서포트, 컨피던스, 리프트를 구해준다. 이때 임계치를 설정하는 파라미터가 있는데, apriori 함수의 min_support옵션과 association_rules 함수의 metric, min_threshold 옵션이다. 패턴분석과 apriori 알고리즘은 동시에 나오는 집합에 집중하며, 미리 설정한 임계치를 넘는 값을 고르는 것이 목적이기 때문에 그 임계치를 설정하는 것이 중요하다. 가능하면 모든 결과를 얻고자 하였으므로 min_support를 0.000001로 설정하여 결과를 출력하였다. 또한 metric은 lift로 인가하고, min_threshold는 0으로 설정하여 연산 결과를 출력하도록 하였다. 또한 연산결과 테이블의 attribute가 antecedents와 consequents로 나눠지는 것을 확인할 수 있다. 원래 각각의 집합들이 antecedents와 consequents 들어간 값이 출력으로 나오는데, 알고자 하는 것이 cardio를 가지는 후행사건에 영향을 주는 집합이기 때문에 consequents가 cardio가 되는 경우만을 설정했다. 예를 들어 age attribute의 연산 결과는 <그림 5>와 같다.

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
6	(60)	(Cardio)	0.229039	0.498388	0.150341	0.656400	1.317048	0.036191	1.459874
3	(50)	(Cardio)	0.507070	0.498388	0.251969	0.496912	0.997039	-0.000748	0.997067
9	(40)	(Cardio)	0.258450	0.498388	0.094899	0.367187	0.736749	-0.033909	0.792670
15	(30)	(Cardio)	0.005442	0.498388	0.001178	0.216524	0.434449	-0.001534	0.640240

<그림 5>

attribute 추출을 위해 임계치를 정해준다. Min_confidence = 0.6, lift = 1로, Min_support = 0.01로 설정하였다. 이때 support의 임계치가 작은 이유는 한 집합의 수가 너무 적은 경우에는 support 값이 자연스럽게 작아지는 것을 고려한 것이다. 위의 <그림 5>를 보면 antecedents가 60 인 경우 모든 기준을 만족하는 것을 확인할 수 있으며, 다른 경우에 비해 연산 결과의 차이가 있음을 확인할 수 있다. 즉, age라는 하나의 attribute에서 cardio라는 consequents와 여러 관계를 발견할 수 있기 때문에 cardio를 잘 설명할 수 있는 attribute 중 하나로 age를 선택하는 것이 좋을 것이라 생각한다. 하지만 이렇게 모든 attribute의 모든 antecedents에 대해서 일일이 분석하는 것이 번거롭기 때문에 plotly 라이브러리를 통해 시각화를 진행하였다. 각 antecedents에 대한 연산값을 찍어주고, 임계치 좌표에 수평선을 그어 기준을 만족하는지 쉽게 확인할 수 있도록 구현했다. 예를 들어 BMI와의 관계에 대해서 HIGH_OBESITY와 OBESITY 가 기준을 만족하는 것을 <그림 6> 를 통해 확인할 수 있다.



<그림 6>

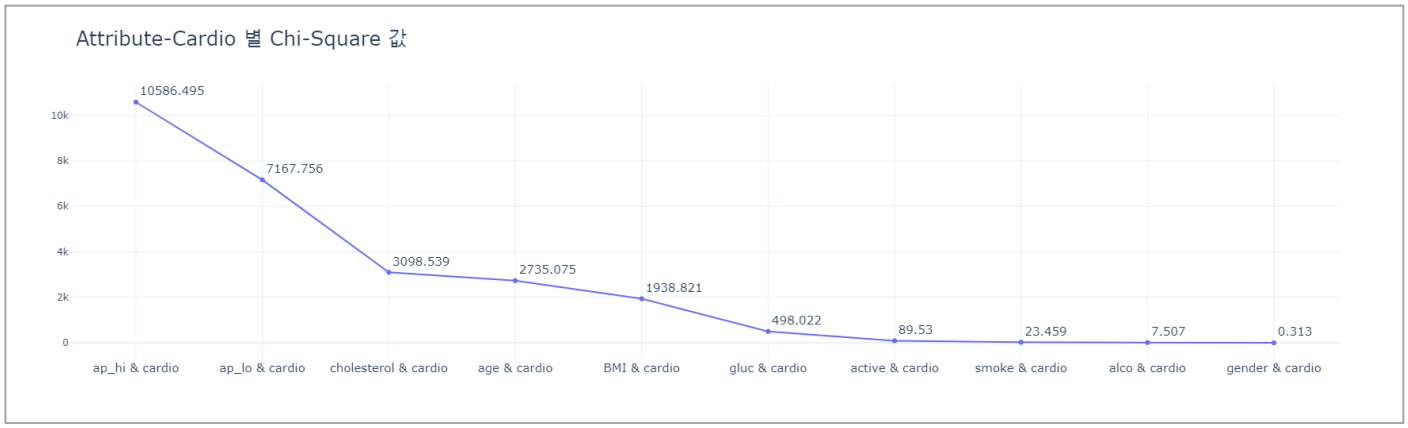
이렇게 모든 attribute의 antecedents에 대해 분석을 통해 추출한 attribute는 <결과 2>와 같다.

<결과 2> : [age, ap_hi, ap_lo, cholesterol, gluc, BMI]

3.3 Null Invariant Metric : kulczynski / Imbalanced Ratio (IR) / chi-square 분석

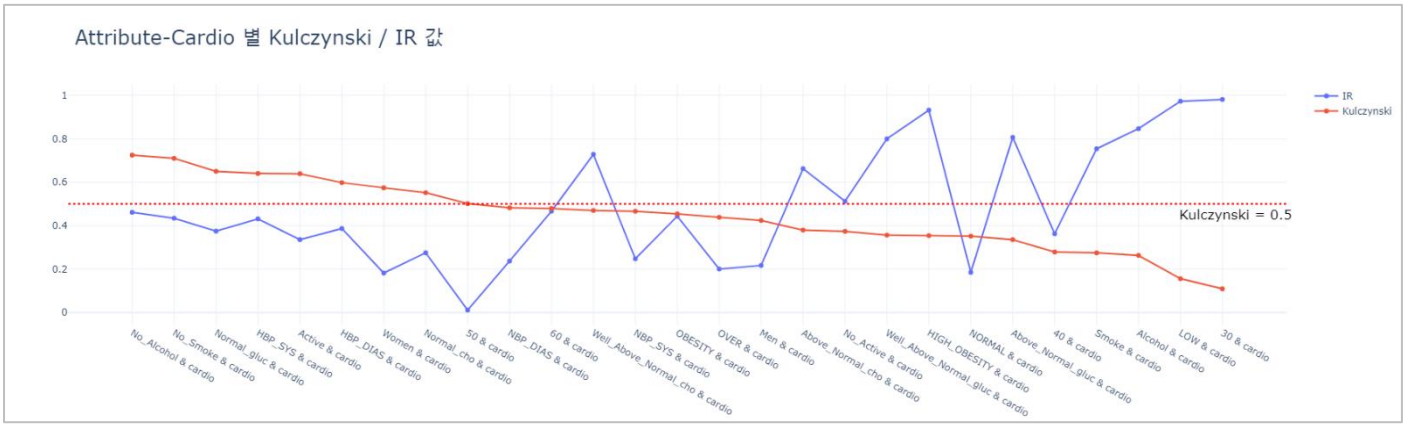
하지만 앞서 진행한 support, confidence, lift는 Null transaction의 영향을 받는다. 따라서 혹시 모를 상황을 방지하기 위해 Null invariant metric에 대해서도 분석을 진행하였다. Kulczynski는 0.5 보다 작으면 negative 관계, 크면 positive 관계임을 의미한다. 하지만 Kulczynski만으로는 정확한 판단을 내릴 수가 없기 때문에 데이터의 분포 형태를 나타내기 위한 imbalanced ratio, 즉 IR을 같이 사용한다. IR이 작을수록 관계에 대한 신빙성이 높아진다. chi square는 그 값이 클수록 집합들 사이의 연관관계가 깊으며, 카이 제곱 분포에 대한 p-value도 구할 수 있다.

.scipy.stats 메서드의 chi2_contingency 라이브러리를 통해 각 attribute에 대한 contingency 테이블을 만들고 이를 바탕으로 각 attribute와 cardio 사이의 chi-square 값을 구해준다. 만들어진 데이터프레임을 통한 결과를 한눈에 볼 수 있도록 시각화 했으며, 대부분 굉장히 큰 chi-square 값을 가지고, smoke attribute부터는 큰 값을 보여주지 못하는 것을 <그림 7>에서 확인할 수 있다.



<그림 7>

다음은 transaction table을 통해 Null Invariant 분석을 진행하였다. chi-square 연산을 위해 contingency 테이블을 만들고, null invariant metric 연산을 위해 support를 계산해주는 함수를 정의해서 각각의 연산식을 구현하고, Jaccard와 kulczynski, IR 값을 chi-square 값과 병합하여 출력했다. 3.2 절의 transaction 데이터에서 볼 수 있듯이, 모든 attribute의 각 case가 총 29개로 나눠지기에 방금 우리가 구한 Jaccard / kulczynski / IR / chi-square / p-value set는 cardio와 no_cardio를 제외한 27개가 나온다. 값들의 편차가 다양하기 때문에 하나의 metric을 기준으로 정렬하여 분석하는 것에도 어려움이 예상되기에 <그림 8>과 같이 시각화를 진행하였다. 가운데에 kulczynski의 기준 값인 0.5 수직선을 표시해주고, kulczynski 와 IR, 그리고 <그림 7>의 chi-square 값을 바탕으로 적절한 attribute를 추출했다.



<그림 8>

예를 들어 30 과 cardio를 보는 경우(<그림 8>의 오른쪽 끝) kulczynski 값은 거의 0에 가깝고, IR은 1에 가깝기 때문에 이 집합들은 서로 negative하며, 두 집합의 분포가 imbalanced 하다는 것을 의미한다. 또한 HBP_SYS와 cardio의 관계를 보면 (<그림 8>의 왼쪽에서 4번째) kulczynski가 0.6보다 크고, IR = 0.4 정도이므로 둘의 관계가 positive하고, 분포 역시 어느 정도 고르다고 말할 수 있다. 따라서 각 case에 대해 IR의 편차가 그지 않은 attribute는 <결과 3-1>과 같고, chi-square에 의한 결과는 <결과 3-2>와 같다. 그리고 <결과 3-1>에서 chi-square 값을 고려해준 결과는 <결과 3>과 같다.

<결과 3-1> : [age, ap_hi, ap_lo, gender, cholesterol, gluc, active]

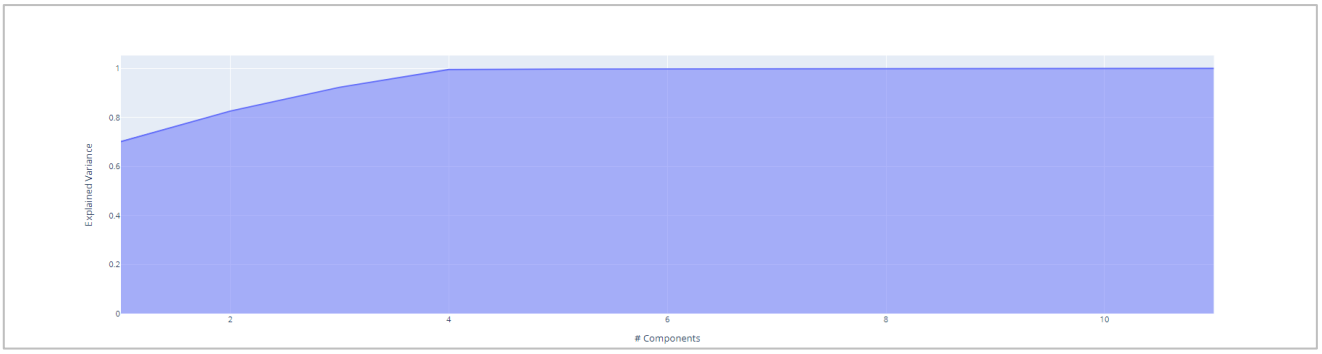
<결과 3-2> : [age, ap_hi, ap_lo, cholesterol, gluc, active, BMI]

<결과 3> : [age, ap_hi, ap_lo, cholesterol, gluc, active, BMI]

3.4 PCA 분석

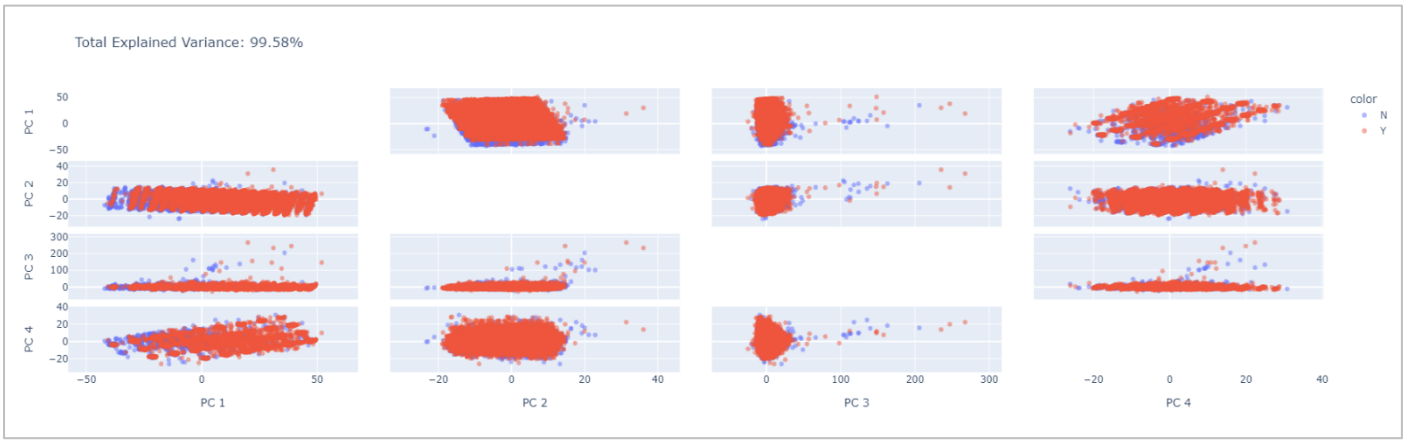
다음으로 sklearn.decomposition모듈의 PCA library 를 사용하여 PCA 분석을 진행했다. PCA의 목적은 기존 attribute들의 선형조합을 통해 서로 orthogonal한 새로운 attribute, 즉 principal components를 만들어 원래 데이터를 새로운 축에서 설명하는 것이다. 원래 데이터의 분산을 가장 잘 설명하는 축부터 PCn (n=1,2,...) 이라 명명하며, 대개 PC1과 PC2를 이용해서 대부분의 variance를 설명할 수 있다. 점점 설명률이 떨어지므로 중간에 축의 선택을 멈추며, 이에 자연스러운 dimensionality reduction이 가능하다. 사실 dimension의 수가 많은 데이터는 아니기 때문에 그렇게 큰 차이는 예상되지 않지만, 데이터로 이것저것 해보는 경험을 얻는 느낌으로 진행해보았다.

우선 기존 데이터를 target과 feature 데이터로 나누었으며, 이를 표준화한 뒤에 principal component로 바꾸는 fit_transfrom 함수를 사용하였다. 그 다음 각 principal component들에 의한 variance의 설명정도를 계산해주는 explained_variance_ratio_를 사용하여 설명정도를 얻을 수 있었다. 사용할 축의 개수를 정하기 위해 개수에 따른 설명정도의 누적합을 나타내는 그래프를 그려보았고, <그림 9>에서 확인할 수 있다. 그 결과 개수가 4개가 된 후부터 설명정도가 거의 1에 가까운 것을 확인했기 때문에, PC4까지 사용하기로 결정하였다.



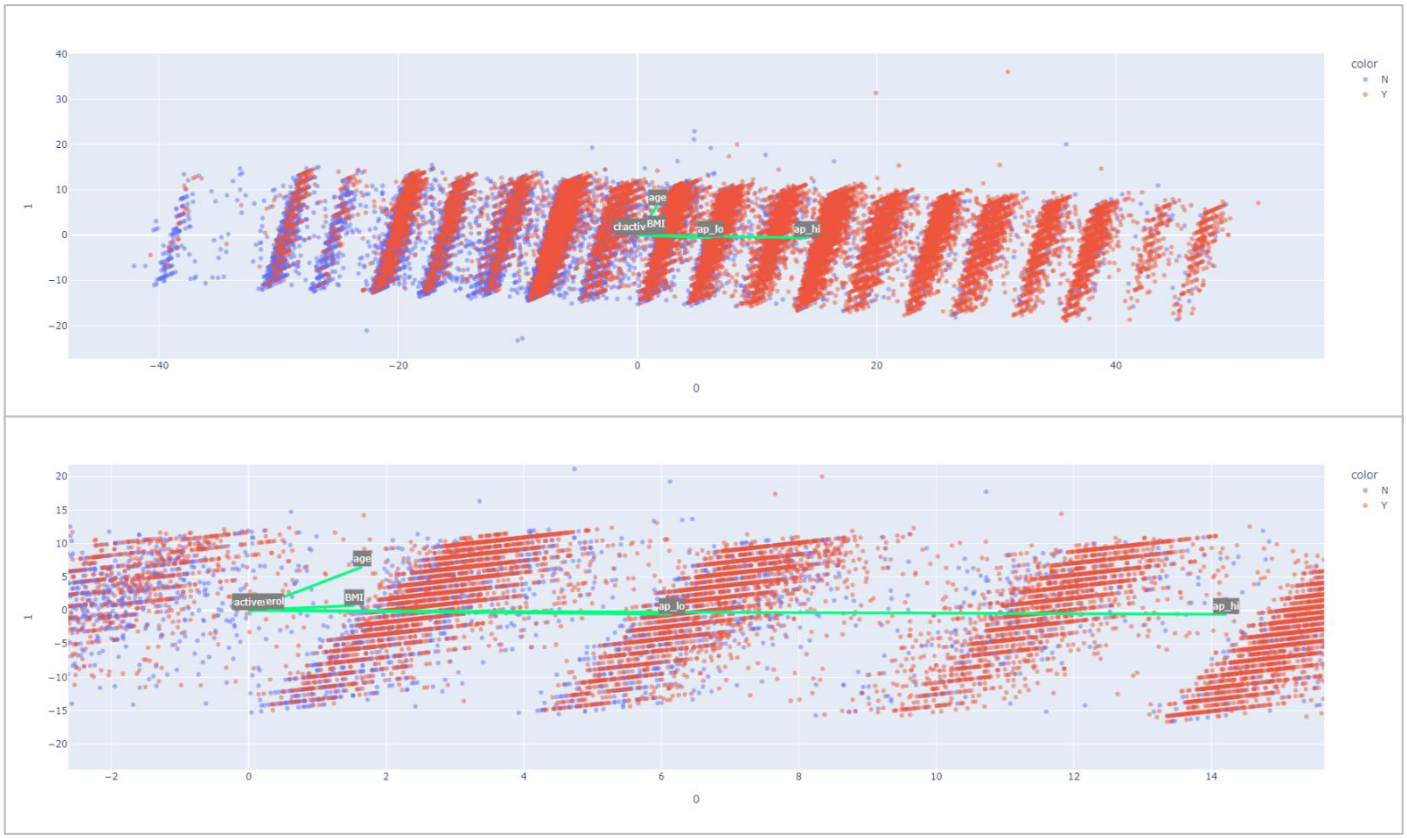
<그림 9>

principal component가 4인 경우의 데이터 분포를 <그림 10>으로 시각화 했다. 이때 파란색 점이 cardio에 걸리지 않은 경우를, 빨간색 점이 걸린 경우를 의미한다. 그 결과 PC1과 PC2가 서로 쌍을 이루는 경우가 다른 경우보다는 좋은 분류를 나타내는 것으로 보인다. 또한 이때의 전체 설명 정도가 99.58% 임을 확인할 수 있었다.



<그림 10>

마지막으로 principal component들을 만들기 위해 사용된 attribute들의 방향성을 볼 수 있게 시각화 했다. plotly에서 제공하는 그래프 확대 기능으로 이 데이터의 분산의 방향성과 서로 비슷한 방향을 가지는, 즉 데이터를 잘 설명해줄 수 있는 attribute들을 비교적 직관적으로 추출하였으며, 이는 <그림 11>에서 확인할 수 있다. 그림상에서는 녹색 선으로 표시되어 있다.



<그림 11>

결론적으로 이 그래프에서는 BMI와 ap-hi, ap_lo attribute가 제일 먼저 눈에 들어오는 같은 방향의 벡터라고 할 수 있다. 이러한 과정을 거쳐서 만들어진 PCA 데이터프레임은 <그림 12>에 나타냈으며, principal component 4까지 사용하였고, 이때의 target은 기존의 데이터와 같은 cardio column을 사용했다.

	PC1	PC2	PC3	PC4	cardio
0	-16.716599	-2.346998	-3.286439	-5.182220	0
1	16.285196	0.514854	5.984210	-1.710913	1
2	-2.086577	-1.087044	-4.772893	11.832023	1
3	27.928691	-8.861411	0.003975	-7.087757	1
5	-5.842537	7.464704	1.720163	-1.162572	0
...	...	...	...	...	...
69994	21.194070	2.804365	-1.505005	10.713512	1
69995	-6.831390	0.265204	0.098980	-0.996545	0
69996	18.542748	9.134694	20.470138	-0.754041	1
69998	7.667006	6.856430	-2.372434	4.471729	1
69999	-6.706255	3.017507	-2.222322	-1.330458	0

64500 rows × 5 columns

<그림 12>

PCA 분석을 통해 추출한 attribute는 <결과 4, 5>와 같다.

<결과 4> : [ap_hi, ap_lo, cholesterol, gluc, BMI]

<결과 5> : [PC1, PC2, PC3, PC4]

3.5 연관관계 분석 결론

이렇게 해서 각각의 metric에 의한 결과 5가지를 얻을 수 있었다. 이 결과들을 통합하여 총 7개의 최종

attribute set을 선택하였으며, 앞으로는 이 attribute를 가지고 Decision Tree와 random forest를 수행하여 결과를 도출할 것이다. 추출할 attribute set은 <표 2>에 나타내었다.

No.	Attribute Set
1	[전체 attribute]
2	[ap_hi, ap_lo, cholesterol, BMI]
3	[age, ap_hi, ap_lo, cholesterol, BMI]
4	[age, ap_hi, ap_lo, cholesterol, gluc, BMI]
5	[age, ap_hi, ap_lo, cholesterol, gluc, active, BMI]
6	[ap_hi, ap_lo, cholesterol, gluc, BMI]
7	[PC1, PC2, PC3, PC4]

<표 2>

4. classification model구현

data가 label을 가지고 있기 때문에 spervised learning을 진행할 것이다. 먼저 decision tree를 선택하였다. Model의 학습과 평가를 위해 training set과 test set이 필요하다. 가지고 있는 data를 test set과 training set으로 나눈다. attribute들이 수치형 자료들과 category가 여러 개인 경우가 있기 때문에 gini대신 entropy를 기준으로 decision tree를 만들었다. 먼저 Attribute를 추출하지 않은, 단순히 전처리만이 이루어진 데이터를 사용했다. Pruning의 효과를 살펴보기 위해 처음에는 트리의 최대 깊이인 max_depth를 None으로 설정했다. 이때의 accuracy는 0.64181이었다.

```
# max_depth = None 인 경우의 train set accuracy
print('Case 1 train Accuracy: %.5f' % accuracy_train(cardio_1_feat, None)[1])
print('Case 2 train Accuracy: %.5f' % accuracy_train(cardio_2_feat, None)[1])
print('Case 3 train Accuracy: %.5f' % accuracy_train(cardio_3_feat, None)[1])
print('Case 4 train Accuracy: %.5f' % accuracy_train(cardio_4_feat, None)[1])
print('Case 5 train Accuracy: %.5f' % accuracy_train(cardio_5_feat, None)[1])
print('Case 6 train Accuracy: %.5f' % accuracy_train(cardio_6_feat, None)[1])
```

```
Case 1 train Accuracy: 0.97181
Case 2 train Accuracy: 0.82638
Case 3 train Accuracy: 0.94109
Case 4 train Accuracy: 0.94673
Case 5 train Accuracy: 0.95736
Case 6 train Accuracy: 0.84334
```

```
# max_depth = None 인 경우의 test set accuracy
print('Case 1 test Accuracy: %.5f' % accuracy_test(cardio_1_feat, None)[1])
print('Case 2 test Accuracy: %.5f' % accuracy_test(cardio_2_feat, None)[1])
print('Case 3 test Accuracy: %.5f' % accuracy_test(cardio_3_feat, None)[1])
print('Case 4 test Accuracy: %.5f' % accuracy_test(cardio_4_feat, None)[1])
print('Case 5 test Accuracy: %.5f' % accuracy_test(cardio_5_feat, None)[1])
print('Case 6 test Accuracy: %.5f' % accuracy_test(cardio_6_feat, None)[1])
```

```
Case 1 test Accuracy: 0.64181
Case 2 test Accuracy: 0.65876
Case 3 test Accuracy: 0.64517
Case 4 test Accuracy: 0.64196
Case 5 test Accuracy: 0.64465
Case 6 test Accuracy: 0.65711
```

<Decision Tree : max_depth = None일 때 train accuracy(좌), test accuracy(우)>

Max_depth = None 일 때 전체적으로 낮은 결과가 나왔고, 이때 training set의 accuracy는 대부분 0.8이나 0.9를 상회했다. 따라서 overfitting이 일어났을 것이라 생각했고, accuracy가 높아지도록 pruning을 해주었다. 최대 깊이를 2에서 15까지 조절하며 accuracy를 측정한 결과 5의 깊이에서 0.72439로 accuracy가 가장 높았다. 하지만 이때는 train set의 accuracy도 많이 낮아졌기에 전체적인 학습능력이 떨어졌을 가능성이 있다.

```
# max_depth = 5 인 경우의 train set accuracy
print('Case 1 train Accuracy: %.5f' % accuracy_train(cardio_1_feat, 5)[1])
print('Case 2 train Accuracy: %.5f' % accuracy_train(cardio_2_feat, 5)[1])
print('Case 3 train Accuracy: %.5f' % accuracy_train(cardio_3_feat, 5)[1])
print('Case 4 train Accuracy: %.5f' % accuracy_train(cardio_4_feat, 5)[1])
print('Case 5 train Accuracy: %.5f' % accuracy_train(cardio_5_feat, 5)[1])
print('Case 6 train Accuracy: %.5f' % accuracy_train(cardio_6_feat, 5)[1])
```

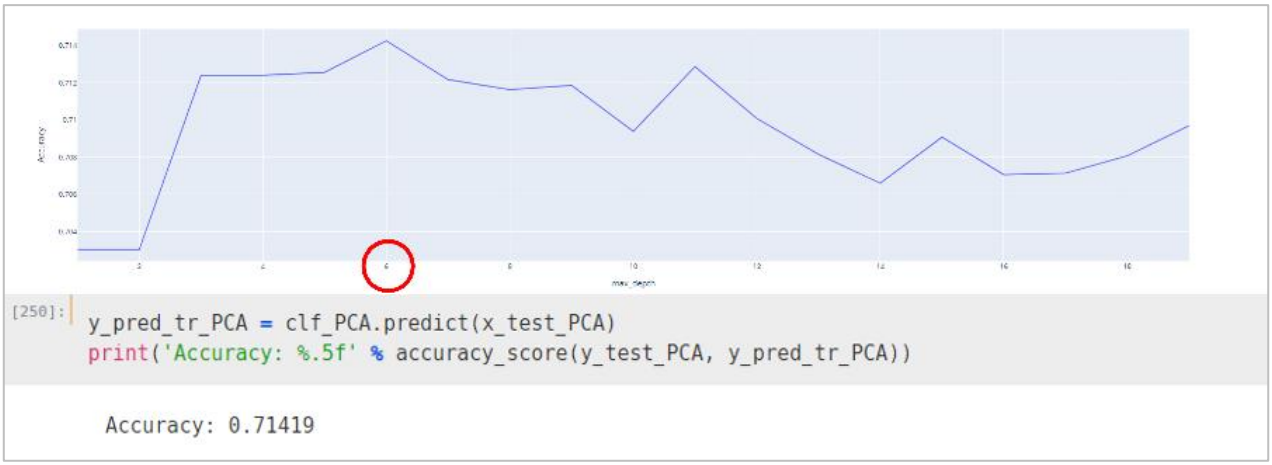
```
Case 1 train Accuracy: 0.72678
Case 2 train Accuracy: 0.72246
Case 3 train Accuracy: 0.72698
Case 4 train Accuracy: 0.72795
Case 5 train Accuracy: 0.72678
Case 6 train Accuracy: 0.72277
```

```
# max_depth = 5 인 경우의 test set accuracy
print('Case 1 test Accuracy: %.5f' % accuracy_test(cardio_1_feat, 5)[1])
print('Case 2 test Accuracy: %.5f' % accuracy_test(cardio_2_feat, 5)[1])
print('Case 3 test Accuracy: %.5f' % accuracy_test(cardio_3_feat, 5)[1])
print('Case 4 test Accuracy: %.5f' % accuracy_test(cardio_4_feat, 5)[1])
print('Case 5 test Accuracy: %.5f' % accuracy_test(cardio_5_feat, 5)[1])
print('Case 6 test Accuracy: %.5f' % accuracy_test(cardio_6_feat, 5)[1])
```

```
Case 1 test Accuracy: 0.72439
Case 2 test Accuracy: 0.71824
Case 3 test Accuracy: 0.72398
Case 4 test Accuracy: 0.72439
Case 5 test Accuracy: 0.72439
Case 6 test Accuracy: 0.71866
```

<Decision Tree : max_depth = 5일 때 train accuracy(좌), test accuracy(우)>

다음은 PCA 결과를 바탕으로 decision tree를 구현하였다. 트리의 깊이가 얼마일때 정확도가 가장 높은지 확인하기 위해 시각화를 했고, 깊이가 6일때 가장 높은 것을 확인할 수 있었다. 그때의 정확도는 0.71419였다.



```
[250]: y_pred_tr_PCA = clf_PCA.predict(x_test_PCA)
       print('Accuracy: %.5f' % accuracy_score(y_test_PCA, y_pred_tr_PCA))
```

```
Accuracy: 0.71419
```

<Decision Tree : max_depth = 6일 때 PCA Dataframe의 accuracy>

model들의 accuracy가 전반적으로 낮았기 때문에 성능을 높이기 위해 ensemble method인 random forest를 사용했다. 깊이는 12, subset은 200개로 나누었을 때 가장 좋은 accuracy인 0.7299를 보여주었다. decision tree에 비해 전체적인 accuracy가 소폭 상승한 것을 확인할 수 있었다.

```
# original data의 모든 attribute를 사용한 경우 : case 1
cardio.target_all = cardio['cardio'].copy()
cardio.feat_all = cardio.drop(['cardio'], axis = 1).copy()

train_x_all, test_x_all, train_y_all, test_y_all = train_test_split(cardio.feat_all, cardio.target_all, test_size=0.3, random_state=0)

# 데이터 표준화 작업
sc = StandardScaler()
sc.fit(train_x_all)

# 표준화된 데이터셋
train_x_std_all = sc.transform(train_x_all)
test_x_std_all = sc.transform(test_x_all)

clf = RandomForestClassifier(random_state=0, max_depth = 12, n_estimators = 200)
clf.fit(train_x_all,train_y_all)

predict4 = clf.predict(train_x_all)
predict3 = clf.predict(test_x_all)

print("Train set accuracy : ", accuracy_score(train_y_all, predict4))
print("Test set accuracy : ", accuracy_score(test_y_all,predict3))
```

```
Train set accuracy :  0.7690586932447397
Test set accuracy :  0.7299741602067183
```

<Random Forest : max_depth = 12일 때 모든 attribute를 사용한 경우의 accuracy>

마지막으로 PCA 결과를 바탕으로 random forest를 구현하였다. 앞서 구한 decision tree accuracy인 0.71419보다 좋은 accuracy를 얻었다. 하지만 train set accuracy와 차이가 많은 것으로 보아 오버피팅을 예상할 수 있었다.

```
clf = RandomForestClassifier(random_state=1)
clf.fit(train_x_PCA,train_y_PCA)

predict2 = clf.predict(train_x_PCA)
predict1 = clf.predict(test_x_PCA)

print("Train set accuracy : ", accuracy_score(train_y_PCA, predict2))
print("Test set accuracy : ", accuracy_score(test_y_PCA,predict1))
```

```
Train set accuracy :  0.9999114064230343
Test set accuracy :  0.7422739018087855
```

<Random Forest : PCA Dataframe의 accuracy>

5. 모델평가 / 결과 해석

원본 데이터 test set의 전체 attribute에 대한 Decision Tree의 accuracy가 max_depth=5 일때 0.72439로 가장 높았다. Accuracy가 생각보다 낮게 나왔기 때문에 오버피팅을 첫번째 염두에 두고 train set에 대한 accuracy를 산출해 보았다. 하지만 max_depth=5 일때 model의 train set에 대한 accuracy 역시 0.726을 조금 상회하는 수준이었고, 이로부터 오버피팅이라기 보다는 학습 자체가 제대로 되지 않았을 가능성이 높다고 생각하였다. 다만, 트리의 최대 깊이를 설정하지 않고 default 상태인 None으로 학습을 진행한 경우에는 test set에서 0.642 정도의 accuracy로 그 성능이 더욱 떨어진 것을 확인하였다. 이에 그 경우의 train set에 대한 accuracy를 구한 결과 대부분의 수치가 0.9를 상회하는 값이 나왔으며, 확실히 오버피팅이 발생한 것을 확인할 수 있었다. 이를 보아 Decision Tree 학습과 테스트에 있어서 Pruning이 중요하다는 것으로 해석할 수 있을 듯 하다. 반면에, 2절의 연관관계 분석을 통해 추출한 attribute를 통한 accuracy는 전체 attribute에 의한 accuracy와 비슷한 수치를 보여주며 연관관계 분석의 유효함을 입증했다. 또한, 이를 Tree들의 집합인 Random Forest에서 분류하였을 때는 max_depth=12 인 경우에 0.72997 로 소폭 상승했다. 여러개의 트리들을 통해 classification을 한 결과로는 그렇게 좋아보이는 상승량은 아니지만, 성능이 조금이나마 향상되는 것을 확인할 수 있었다.

다음으로는 PCA 분석을 통해 얻은 principal component들을 사용하여 classification을 진행하였다. 우선 Decision Tree의 경우에는 max_depth=6에서 가장 높은 accuracy를 가졌으며, 그때의 결과는 0.714 정도였다. 따라서 이 경우에도 오버피팅을 생각하고 train set에 대한 accuracy를 산출했으나, 그 경우의 accuracy 역시 0.72를 조금 상회하는 결과가 나왔다. 따라서 이 경우에도 학습이 제대로 이루어지지 않았다는 결론을 내릴 수 있었다. 또한 max_depth를 None로 설정했을 때에는 train set에 대한 accuracy가 0.99로 완전히 학습에 성공한 반면, test set accuracy는 0.71 정도로 오히려 성능이 살짝 떨어졌다. 이 경우 역시 오버피팅이 발생한 것으로 사료되며, pruning의 중요성을 입증한 경우라고 볼 수 있을 것 같다. 이어서 진행한 Random Forest에서는 test set에 대한 accuracy가 0.742 정도로 원래 데이터를 사용해서 Random Forest를 수행했을 때보다 높은 성능의 향상을 보여주었다. 하지만 이때의 train set에 대한 accuracy가 0.99가 나옴으로써 오버피팅이 일어났다고 결론을 내렸다.

일반적으로 우리는 test set에 대한 accuracy가 높은 모델을 선호할 수 밖에 없다. 아무래도 데이터를 입력했을 때 가장 먼저 눈에 보이는 지표이고, 정답을 많이 예측하는 모델이 우리가 만들기를 원하는 모델이기 때문이다. 이러한 통념 하에서는 이번 프로젝트에 있어 PCA 분석을 통한 Random Forest 모델이 0.742 정도의 accuracy로 가장 좋은 모델이라고 말할 것이다. 하지만 이는 모델이 train set에 대해 오버피팅이 일어났다는 사실을 고려하지 않은 결론이다. 물론 오버피팅이 있더라도 test set의 결과가 눈에 띄게 좋다면 그 모델을 고르는 것이 맞겠지만, 각각의 accuracy들은 분포가 거의 균일한 값들을 가진다. 따라서, 바라던 만큼의 좋은 결과가 나오지는 않았으나 위에서 언급한 모델에서 하나를 고른다면 연관관계 분석을 통해 추출한 attribute set (5) (<표 2> 참고) 를 사용하여 Random Forest를 수행하였거나, 원본 데이터의 전체 attribute를 가지고 Random Forest를 수행한 모델을 선택하는 것이 최선의 선택이 될 것이다. Accuracy가 그렇게 높게 나오지 않은 이유는 아무래도 object는 많지만 target의 분포가 balanced하고, dimension이 큰 데이터가 아니기 때문일 것이다. 즉, target의 균일한 분포를 설명하기에는 dimension이 부족했다는 의미이다. 따라서 굳이 dimensionality reduction을 하지 않아도 모델을 학습시키는 데 무리가 없는, 즉 원본 그 상태로도 정보가 부족했을 가능성이 있는 데이터이지 않을까 생각한다.

6. 느낀점

수업시간에 배운 다양한 연산들을 실제 코드로 구현하는 것이 재밌었다. 또한 데이터의 형태만 맞춰주면 알고 싶은 값을 반환해주는 것을 보고 함수의 내부 구조가 궁금해지기도 했다. 특히 단순히 결과를 구하는 것에 그치는 것이 아니라 결과들을 바탕으로 그들간의 관계를 직접 알아내는 경험을 해본 것이 이 프로젝트에서 얻은 가장 큰 수확이라고 생각한다. 이렇게 체계적으로 데이터를 분석해본 것이 처음이라서 전체적으로 미숙하고, 정해진 루트가 있다기보다는 실제로 사용해야할 방법들을 찾아가면서 프로젝트를 수행하는 것이 어려웠다. 하지만 대학교에 와서도 이론적인 공부에 지쳐있는 우리에게는 좋은 경험이었다고 생각한다.