2023 DATA·AI 분석 경진대회 과학기술 부문

사상체질 판별 모델 개발

Team 데마시아

김현동 한진규 김세원

목차

I. 프로젝트 개요

Team 데마시아

김현동 한진규 김세원

- 연세대학교 산업공학과 재학
- 다수의 '겹강' 및 팀 프로젝트로 다져진 전우애
- 데이터마이닝을 줄인 '데마'에서 시작해 '리그오브레전드'의 가상 국가 중 하나인 '데마시아'까지

I. 프로젝트 개요

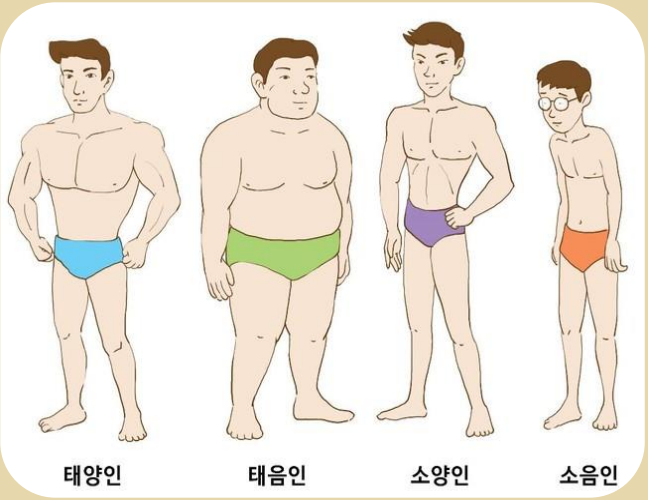
- 한의학은 현대의학과 달리 사람들의 체질과 오장육부의 기능 강약 차이 등을 근거로 삼으며 질병 원인을 규명함

- 그러나 환자에게 있어 '신뢰'를 주지 못한다는 평가를 받고 있으며, 지난해 공개된 '한의약산업실태조사'에 따르면 향후 성장 필요 요인 1순위는 '고객과의 신뢰 구축'

- 사상체질은 체형기상, 용모사기, 성질재간, 병증약리를 바탕으로 사람의 체질을 태양인, 소양인, 태음인, 소음인으로 구분해놓은 것

- 한의학 처방 및 치료의 기본이 되는 사상체질 분류를 데이터 분석적 접근을 통해 보다 객관적인 결과를 제공하고자 본 프로젝트 기획

- 치료에서의 가치 뿐만 아닌, 고객과의 신뢰 구축 및 데이터 시대에서 한의학이 발전할 토대가 될 것이라 생각



https://m.locallife.news/news/view/1065617844853689

II. 활용 데이터

- 한국한의학연구원에서 제공한 500명의 체질 데이터 활용 → 태양인(실제 표본 수 ↓)제외 3개의 체질로 분류하는 Classification Problem

- 성별, 나이, 결혼 여부, 몸무게 등 기존 사상체질 분류에 사용되는 총 71개의 특성

	ID	SEX	AGE	JOB	EDUCATION	MARRIAGE	DRINK	SMOKE	CHARACTER1	CHARACTER2	...	BODYMEAS_8HAND6	BODYMEAS_8HAND7	BODYMEAS_8HAND8	SYSTOLICBP	DIASTOLICBP	GLUCOSE
0	KDCT00001	2	48.666	2	5	2	NaN	NaN	2	2	...	82.0	91.0	96.0	120	80	104.0
1	KDCT00002	2	80.773	14	3	2	NaN	NaN	1	1	...	92.0	96.0	99.0	124	65	110.0
2	KDCT00003	1	50.008	3	3	2	NaN	NaN	3	1	...	80.0	81.0	81.0	121	75	79.0
3	KDCT00004	2	50.551	14	4	2	NaN	NaN	1	1	...	91.0	94.0	99.0	113	81	100.0
4	KDCT00005	1	49.581	8	2	2	NaN	NaN	3	2	...	98.0	98.0	100.0	129	72	92.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
495	KDCT00496	1	50.682	3	5	2	1.0	2.0	3	1	...	74.5	80.0	88.0	120	80	71.0
496	KDCT00497	2	42.542	3	5	2	2.0	3.0	3	1	...	64.5	76.0	79.0	110	70	74.0
497	KDCT00498	1	48.797	7	5	2	1.0	2.0	3	2	...	95.0	96.5	97.5	150	100	125.0
498	KDCT00499	1	51.263	7	4	2	1.0	1.0	3	2	...	86.0	89.5	91.4	110	80	85.0
499	KDCT00500	2	53.767	5	3	2	3.0	3.0	2	1	...	76.2	87.3	91.0	120	80	84.0

500 rows × 73 columns

주어진 데이터의 DataFrame

III. 모델 개발 방법

데이터 전처리 – 결측치 처리

- 주어진 데이터에서 결측치를 가지는 column이 많았기에 결측치 삭제 / 보간 방법 구상

Data columns (total 73 columns):

#	Column	Non-Null Count	Dtype
0	ID	500 non-null	object
1	SEX	500 non-null	int64
2	AGE	500 non-null	float64
3	JOB	500 non-null	int64
4	EDUCATION	500 non-null	int64
5	MARRIAGE	500 non-null	int64
6	DRINK	26 non-null	float64
7	SMOKE	26 non-null	float64
8	CHARACTER1	500 non-null	int64
9	CHARACTER2	500 non-null	int64
10	CHARACTER3	500 non-null	int64
11	CHARACTER4	500 non-null	int64
12	CHARACTER5	500 non-null	int64
13	CHARACTER6	500 non-null	int64
14	CHARACTER7	499 non-null	float64
15	CHARACTER8	499 non-null	float64
16	CHARACTER9	500 non-null	int64
17	CHARACTER10	500 non-null	int64
18	CHARACTER11	500 non-null	int64
19	CHARACTER12	500 non-null	int64
20	CHARACTER13	500 non-null	int64
21	CHARACTER14	500 non-null	int64
22	CHARACTER15	500 non-null	int64
23	DIET4	421 non-null	float64
24	DIET7	474 non-null	float64
25	DIGEST1	500 non-null	int64
26	DIGEST3	500 non-null	int64
27	SWEAT1	500 non-null	int64
28	SWEAT3	498 non-null	float64
29	STOOL1	500 non-null	int64
30	STOOL7	500 non-null	int64
31	STOOL12_7	267 non-null	float64
32	STOOL12_8	267 non-null	float64
33	URINE2	500 non-null	float64
34	COLDHEAT1	500 non-null	int64
35	COLDHEAT2	500 non-null	int64
36	COLDHEAT3	500 non-null	int64
37	COLDHEAT4	500 non-null	int64
38	WATER1	499 non-null	float64
39	WATER3	500 non-null	int64
40	COLDHEAT_S	499 non-null	float64
41	COLDHEAT_G	499 non-null	float64
42	HYPER1	500 non-null	int64
43	DIABE1	500 non-null	int64
44	HYPERLI1	500 non-null	int64
45	HEALTH1	499 non-null	float64
46	SLEEP3_1	500 non-null	int64
47	SLEEP3_2	500 non-null	int64
48	FATIGUE1	500 non-null	int64
49	FATIGUE2_1	500 non-null	int64
50	FATIGUE2_2	500 non-null	int64
51	FATIGUE2_3	500 non-null	int64
52	FATIGUE2_4	500 non-null	int64
53	FATIGUE2_5	500 non-null	int64
54	FINALDIAGNOSIS	500 non-null	int64
55	HEIGHT	500 non-null	float64
56	WEIGHT	500 non-null	float64
57	BMI	500 non-null	float64
58	BODYMEAS_8HAND1	500 non-null	float64
59	BODYMEAS_8HAND2	500 non-null	float64
60	BODYMEAS_8HAND3	500 non-null	float64
61	BODYMEAS_8HAND4	500 non-null	float64
62	BODYMEAS_8HAND5	500 non-null	float64
63	BODYMEAS_8HAND6	500 non-null	float64
64	BODYMEAS_8HAND7	500 non-null	float64
65	BODYMEAS_8HAND8	500 non-null	float64
66	SYSTOLICBP	500 non-null	int64
67	DIASTOLICBP	500 non-null	int64
68	GLUCOSE	497 non-null	float64
69	T_CHOL	497 non-null	float64
70	TG	497 non-null	float64
71	HDL_CHOL	497 non-null	float64
72	LDL_CHOL	497 non-null	float64

- ‘DRINK’, ‘SMOKE’는 오직 26명이 답변했고, 그 외에도 ‘STOOL12_7’, ‘STOOL12_8’에 다수의 결측치 존재

- 결측치를 1개 이상 포함하는 특성 = ['DRINK', 'SMOKE', 'CHARACTER7', 'CHARACTER8', 'DIET4', 'DIET7', 'SWEAT3', 'STOOL12_7', 'STOOL12_8', 'WATER1', 'COLDHEAT_S', 'COLDHEAT_G', 'HEALTH1', 'GLUCOSE', 'T_CHOL', 'TG', 'HDL_CHOL', 'LDL_CHOL']

III. 모델 개발 방법

데이터 전처리 – 결측치 처리

- 결측치 보간을 수행하기 위해 결측된 자리에 모든 응답자들의 해당 특성 평균(mean)치 사용
- 다만 결측치도 특성 별 상이함이 존재 → 결측치가 소수인 경우, 해당 특성과 연관이 많은 기타 특성의 분포를 고려해서 보간

e.g. DIET4, DIET7의 결측치를 채우기 위해 DIGEST1, DIGEST3의 데이터를 활용

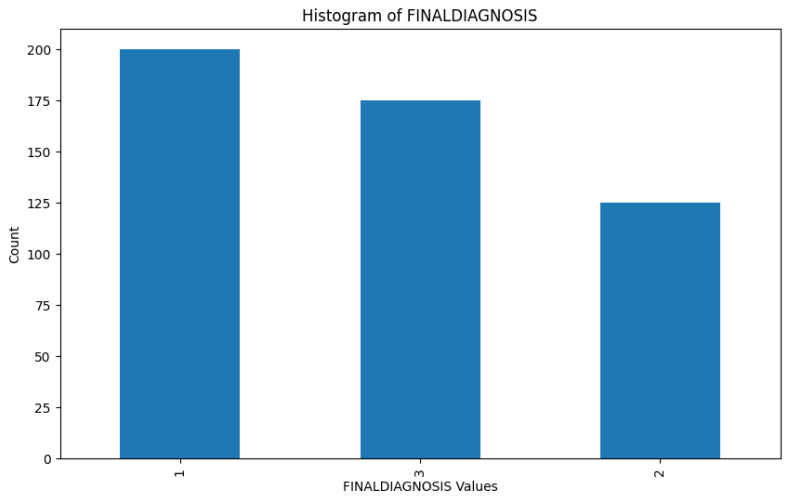
#	Column	Non-Null Count	Dtype
0	SEX	500 non-null	int64
1	AGE	500 non-null	float64
2	JOB	500 non-null	int64
3	EDUCATION	500 non-null	int64
4	MARRIAGE	500 non-null	int64
5	DRINK	500 non-null	float64
6	SMOKE	500 non-null	float64
7	CHARACTER1	500 non-null	int64
8	CHARACTER2	500 non-null	int64
9	CHARACTER3	500 non-null	int64
10	CHARACTER4	500 non-null	int64
11	CHARACTER5	500 non-null	int64
12	CHARACTER6	500 non-null	int64
13	CHARACTER7	500 non-null	float64
14	CHARACTER8	500 non-null	float64
15	CHARACTER9	500 non-null	int64
16	CHARACTER10	500 non-null	int64
17	CHARACTER11	500 non-null	int64
18	CHARACTER12	500 non-null	int64
19	CHARACTER13	500 non-null	int64
20	CHARACTER14	500 non-null	int64
21	CHARACTER15	500 non-null	int64
22	DIET4	500 non-null	float64
23	DIET7	500 non-null	float64
24	DIGEST1	500 non-null	int64
25	DIGEST3	500 non-null	int64
25	DIGEST3	500 non-null	int64
26	SWEAT1	500 non-null	int64
27	SWEAT3	500 non-null	float64
28	STOOL1	500 non-null	int64
29	STOOL7	500 non-null	int64
30	STOOL12_7	500 non-null	float64
31	STOOL12_8	500 non-null	float64
32	URINE2	500 non-null	float64
33	COLDHEAT1	500 non-null	int64
34	COLDHEAT2	500 non-null	int64
35	COLDHEAT3	500 non-null	int64
36	COLDHEAT4	500 non-null	int64
37	WATER1	500 non-null	float64
38	WATER3	500 non-null	int64
39	COLDHEAT_S	500 non-null	float64
40	COLDHEAT_G	500 non-null	float64
41	HYPER1	500 non-null	int64
42	DIABE1	500 non-null	int64
43	HYPERLI1	500 non-null	int64
44	HEALTH1	500 non-null	float64
45	SLEEP3_1	500 non-null	int64
46	SLEEP3_2	500 non-null	int64
47	FATIGUE1	500 non-null	int64
48	FATIGUE2_1	500 non-null	int64
49	FATIGUE2_2	500 non-null	int64
50	FATIGUE2_3	500 non-null	int64
51	FATIGUE2_4	500 non-null	int64
52	FATIGUE2_5	500 non-null	int64
53	FINALDIAGNOSIS	500 non-null	int64
54	HEIGHT	500 non-null	float64
55	WEIGHT	500 non-null	float64
56	BMI	500 non-null	float64
57	BODYMEAS_8HAND1	500 non-null	float64
58	BODYMEAS_8HAND2	500 non-null	float64
59	BODYMEAS_8HAND3	500 non-null	float64
60	BODYMEAS_8HAND4	500 non-null	float64
61	BODYMEAS_8HAND5	500 non-null	float64
62	BODYMEAS_8HAND6	500 non-null	float64
63	BODYMEAS_8HAND7	500 non-null	float64
64	BODYMEAS_8HAND8	500 non-null	float64
65	SYSTOLICBP	500 non-null	int64
66	DIASTOLICBP	500 non-null	int64
67	GLUCOSE	500 non-null	float64
68	T_CHOL	500 non-null	float64
69	TG	500 non-null	float64
70	HDL_CHOL	500 non-null	float64
71	LDL_CHOL	500 non-null	float64

→ 결측치 보간 후 500명 모두 특성 값을 모두 가지는 데이터셋 완성

III. 모델 개발 방법

데이터 전처리 – 클래스 불균형

- 주어진 데이터셋에서 태음인, 소음인, 소양인 데이터의 클래스 수가 불균형적 → Accuracy Paradox 야기할 수 있음
- Train Data에 SMOTE를 이용해서 데이터를 증강함



원본 데이터 클래스 균형

```
y_train.value_counts()

FINALDIAGNOSIS
0    152
2    146
1    102
Name: count, dtype: int64
```

```
x_shuffled = sklearn.utils.shuffle(X_train, random_state=312)
y_shuffled = sklearn.utils.shuffle(y_train, random_state=312)

smote = SMOTE(random_state=312)
X_train, y_train = smote.fit_resample(x_shuffled, y_shuffled)
```

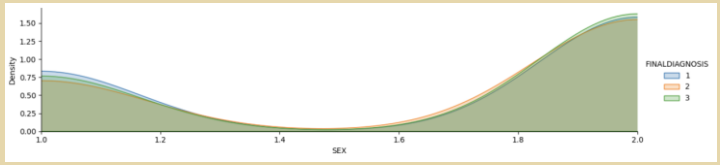
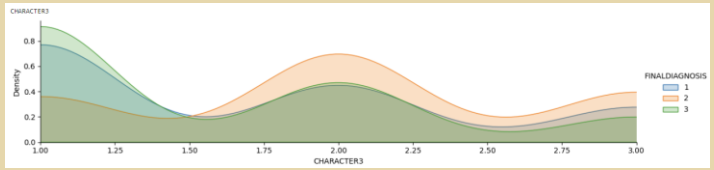
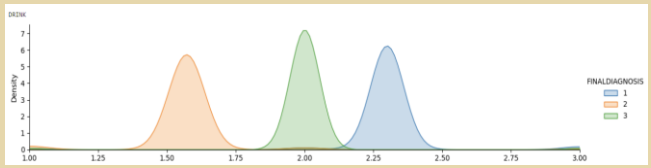
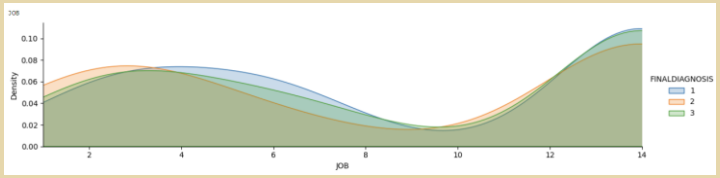
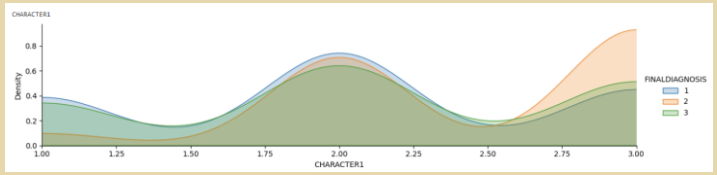
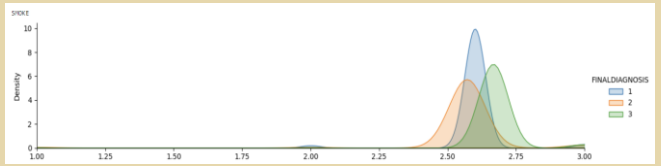
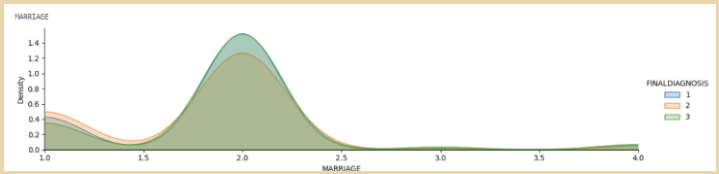
```
y_train.value_counts()

FINALDIAGNOSIS
2    152
0    152
1    152
Name: count, dtype: int64
```

Train Data 클래스 균형

III. 모델 개발 방법

데이터 전처리 – 체질 별 특성 분포 파악

- Seaborn의 facet_grid를 이용해서 체질별로 특성들이 어떻게 분포되어 있는지 확인

설명 근거 LOW	설명력 존재	추가적 고려 필요
 SEX	 CHARACTER 3 (적극/소극)	 DRINK
 JOB	 CHARACTER 1 (대범, 섬세)	 SMOKE
 MARRIAGE		

III. 모델 개발 방법

데이터 전처리 – 입력 변수 선택

- ANOVA를 이용해 클래스 별 데이터의 분포 차이가 유의미($\alpha = 0.05$)하면 보존함
- ID 등 출력변수와 무관한 데이터 삭제, 성능 악화 변수 (DRINK, SMOKE) 삭제
- 최종적으로 37개의 입력 변수 선택

```
kill = []
for col in df.columns:
    f_statistic, p_value = stats.f_oneway(df1[col], df2[col], df3[col])
    alpha = 0.05  # 유의수준 (보통 0.05 사용)
    if p_value > alpha:
        kill.append(col)
    print(col, p_value)
    print()
```

```
SEX 0.7962500816698056
```

```
X
```

	AGE	EDUCATION	CHARACTER1	CHARACTER2	CHARACTER3	CHARACTER6	CHARACTER7	CHARACTER9
0	48.666	5	2	2	1	1	1.0	2
1	80.773	3	1	1	1	1	1.0	1
2	50.008	3	3	1	1	1	1.0	1
3	50.551	4	1	1	1	1	1.0	1
4	49.581	2	3	2	3	3	2.0	2
...	...	...	...	...	...	...	...	...
495	50.682	5	3	1	2	2	2.0	2
496	42.542	5	3	1	3	3	3.0	3
497	48.797	5	3	2	2	2	3.0	2
498	51.263	4	3	2	1	1	1.0	1
499	53.767	3	2	1	1	2	1.0	2

500 rows × 37 columns

- Tree-based Classifier (특히 XGBoost)에서는 오히려 변수 차원을 축소하고 모델을 학습했을 때 좋지 못한 성능 도출 → DRINK & SMOKE 제거한 입력 변수 유지
- ANN은 중요도 낮은 입력변수의 계수를 0으로 축소시키는 알고리즘

```
SVC 0.4737
DTC 0.9737
RFC 0.9847
GNB 0.6534
MLP 0.5485
XGB 0.9825
ETC 0.9583
```

III. 모델 개발 방법

다양한 모델 적용

적용 데이터셋 정리:

- ANN : 69개의 입력변수, test 데이터 증강
- KNN : 40개의 입력변수, test 데이터 증강
- Tree-based Algorithms: 69개의 입력변수, test 데이터 증강

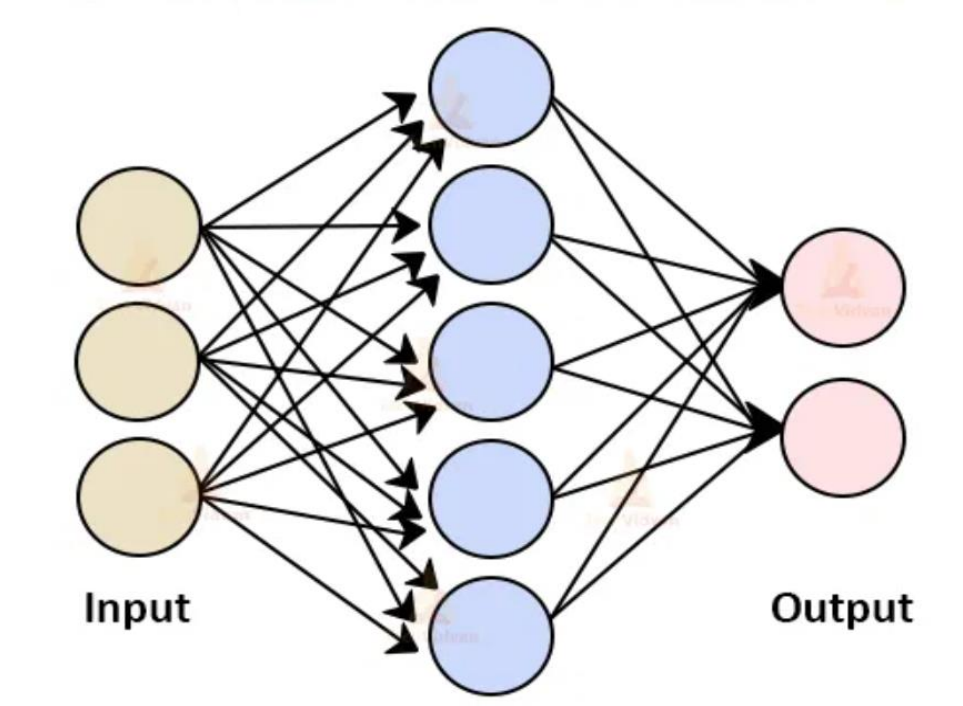
- Classification problem에서 사용하는 대표적인 알고리즘인 ANN Classification, K-NN Classification, Decision Tree Classification, Random Forest Classification, XGB Classifier, Extra Trees Classifier에 중점
- Gaussion Naïve Bayes, Support Vector Machine 등도 이용했으나 성능 ↓

- Classification Algorithm별 성능지표는 f1 score와 accuracy를 사용했으며, f1 score는 Precision과 Recall의 조화평균으로 유의미한 성능지표로 사용될 수 있음

IV. 실험 및 평가

ANN Classification

- ANN Architecture



https://www.analyticsvidhya.com/blog/2021/07/understanding-the-basics-of-artificial-neural-network-ann/

- Input layer, hidden layer, output layer 존재
- Hidden layer 수와 노드 개수는 hyperparameter
- Layer 사이에 활성화 함수 존재 → 선형 연산 값을 비선형으로 변환

IV. 실험 및 평가

ANN Classification

```
# Splitting data into 80% training and 20% testing and validation (10% each)
X_train, X_temp, y_train, y_temp = train_test_split(data_scale, y, test_size=0.2, random_state=42, stratify = y)

# Convert the arrays to PyTorch tensors
X_train = torch.tensor(X_train.values, dtype=torch.float32)
y_train = torch.tensor(y_train.values, dtype=torch.long)
X_temp = torch.tensor(X_temp.values, dtype=torch.float32)
y_temp = torch.tensor(y_temp.values, dtype=torch.long)

# Splitting the remaining 20% into 50% testing and 50% validation
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify = y_temp)
```

```
class Model(nn.Module):
    def __init__(self, in_features = 69, h1 = 120, h2 = 60, h3 = 30, h4 = 15, h5 = 8, out_features = 4):
        super().__init__()

        self.fc1 = nn.Linear(in_features, h1)
        self.fc2 = nn.Linear(h1, h2)
        self.fc3 = nn.Linear(h2, h3)
        self.fc4 = nn.Linear(h3, h4)
        self.fc5 = nn.Linear(h4, h5)
        self.out = nn.Linear(h5, out_features)

    def forward(self,x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = F.relu(self.fc5(x))
        x = self.out(x)

        return x
```

- 69개의 입력 변수
- Hidden Layer 1: 120개 노드
- Hidden Layer 2: 60개 노드
- Hidden Layer 3: 30개 노드
- Hidden Layer 4: 15개 노드
- Hidden Layer 5: 8개 노드
- 활성화 함수는 ReLu 사용

- Train : Test = 8: 2, test 데이터의 절반은 Validation 데이터로 활용

IV. 실험 및 평가

ANN Classification

```
epoch 500, Training Loss: 2.632936229929328e-06, Validation Loss: 13.440617561340332
epoch 500, Training ACC: 1.0, Validation Acc: 0.5

555 55
11.308338165283203
```

```
X_test = StandardScaler().fit_transform(X_test)
```

```
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = y_test
```

```
model = Model()
model.load_state_dict(torch.load('best_model.pth'))
model.eval()
```

```
Model(
  (fc1): Linear(in_features=69, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=60, bias=True)
  (fc3): Linear(in_features=60, out_features=30, bias=True)
  (fc4): Linear(in_features=30, out_features=15, bias=True)
  (fc5): Linear(in_features=15, out_features=8, bias=True)
  (out): Linear(in_features=8, out_features=4, bias=True)
)
```

```
correct_predictions = (predicted_labels == y_test_tensor).sum().item()
total_samples = y_test_tensor.size(0)
accuracy = (correct_predictions / total_samples) * 100

print(f'Model Accuracy on Test Data: {accuracy:.2f}%')
```

```
Model Accuracy on Test Data: 56.00%
```

- 에포크 = 100,000
- 손실 함수 = Cross Entropy
- Optimizer = Adam (learning rate = 0.01)

→ 테스트 데이터에 대한
예측 성능 (Accuracy) = 56%

IV. 실험 및 평가

Before Moving On

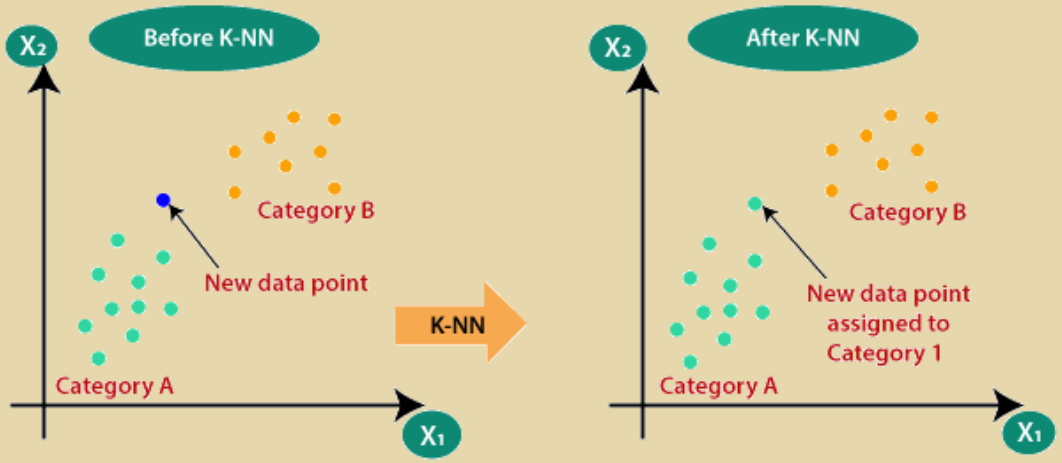
DRINK와 SMOKE를 제거한 데이터셋을 이용해서 hyperparameter 튜닝 과정 없이 다양한 알고리즘을 적용한 raw 결과:

```
SVC 0.4259
DTC 0.6167
RFC 0.7071
GNB 0.521
MLP 0.5041
XGB 0.8054
ETC 0.6532
```

→ hyperparameter 튜닝을 통해 알고리즘 별 성능 개선 및 최적의 hyperparameter 탐색을 목적으로 함

IV. 실험 및 평가

K-NN Classification



https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

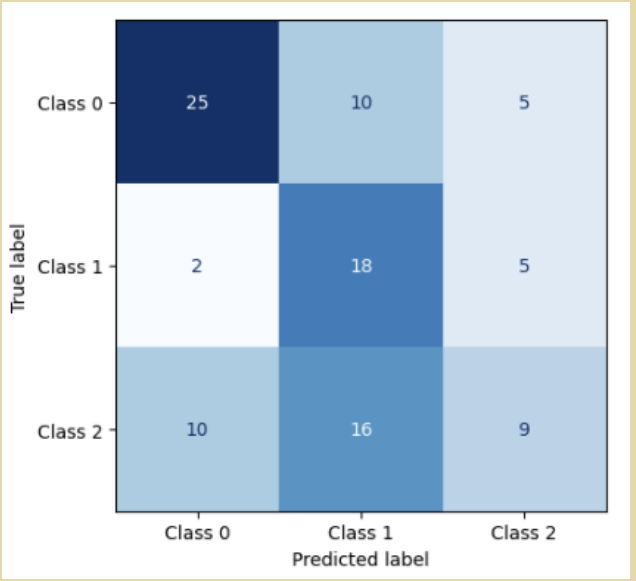
- K 개의 '이웃' 데이터를 기반으로 새로운 데이터의 분포를 예측
- 고려할 주변 데이터 수 , k(n)는 hyperparameter
- 입력 변수 별 scale이 큰 영향을 주기에, 데이터 scaling 필수

IV. 실험 및 평가

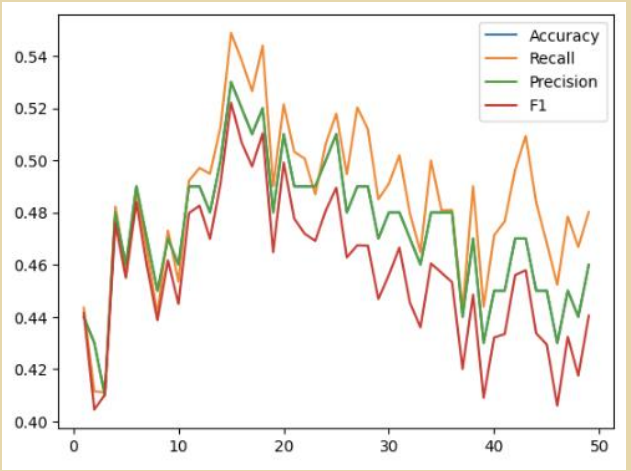
K-NN Classification

```
# Provide labels for all three classes ('Class 0', 'Class 1', 'Class 2')
class_labels = ['Class 0', 'Class 1', 'Class 2']
cm = confusion_matrix(test_y, pred)

# Display confusion matrix with appropriate class labels
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_labels
disp.plot(cmap='Blues', colorbar=False)
```

```
Accuracy: 0.520
Recall Score: 0.520
Precision Score: 0.538
F1 Score: 0.507
```



Resulting Confusion Matrix

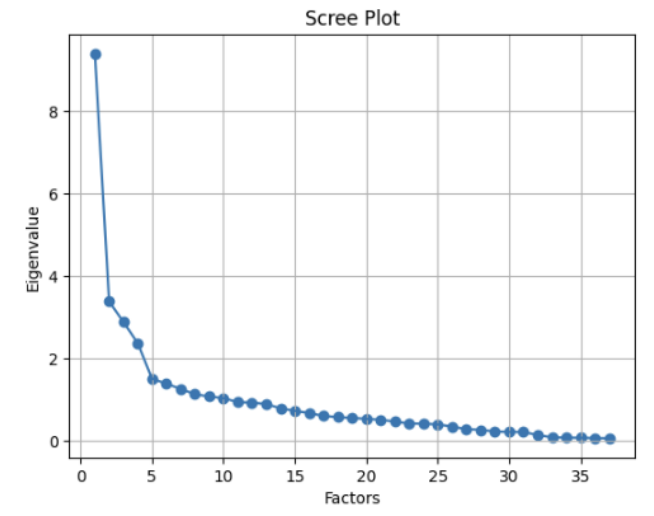


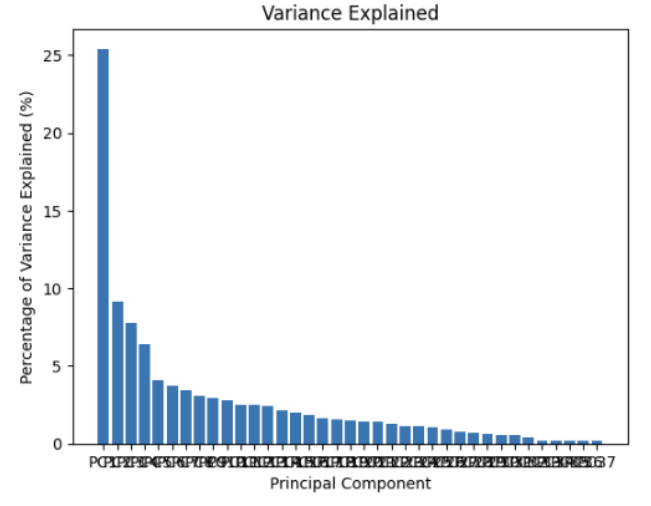
n_neighbor decision

- Train : Test = 8 : 2
- Accuracy = 0.520 , F1 Score = 0.507

IV. 실험 및 평가

K-NN Classification + PCA






```
분산 설명 비율:
 [0.25409871 0.09151851 0.07804152 0.06415241 0.0408582  0.03755308
 0.03414491 0.03063064 0.02915106 0.0280603  0.02543744 0.02513854
 0.02415101 0.02138579 0.01968768 0.01828294 0.0163748  0.01564791
 0.01504745 0.01447524 0.01385089 0.01267528 0.01140054 0.01124646
 0.010868   0.00938932 0.00774133 0.00725045 0.00619085 0.00585136
 0.00581529 0.00387756 0.00230074 0.00210917 0.00201858 0.00188795
 0.00168806]
누적 분산 설명 비율:
 [0.25409871 0.34561722 0.42365874 0.48781115 0.52866935 0.56622243
 0.60036734 0.63099798 0.66014904 0.68820934 0.71364678 0.73878532
 0.76293634 0.78432213 0.80400981 0.82229275 0.83866755 0.85431547
 0.86936292 0.88383816 0.89768905 0.91036433 0.92176486 0.93301132
 0.94387932 0.95326864 0.96100997 0.96826042 0.97445128 0.98030263
 0.98611793 0.98999549 0.99229623 0.9944054  0.99642399 0.99831194
 1.        ]

d = np.argmax(cumsum >= 0.80) +1 #누적된 설명가능한 분산의 비율이 0.80 이상
print('적합한 PC 개수:', d)

적합한 PC 개수: 15
```

	PC0	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC14
0	-237.745184	-94.592008	-28.116916	-16.789193	2.176456	34.953076	88.624697	-36.882782	51.303975	4.217045	20.805760	49.271046	-2.763311	58.038836	39.961704
1	-261.329006	-103.242420	-34.625276	-30.557873	-7.158628	31.935358	71.664074	-28.333769	55.015689	-2.099041	19.570668	45.647357	9.148572	65.351533	40.820274
2	-232.257455	-91.044864	-24.973369	-15.387523	14.746423	30.837054	76.063899	-20.896779	53.243769	-5.231041	12.926144	38.523000	6.255342	57.905205	36.208213
3	-267.091613	-100.270349	-29.283245	-19.225482	0.297863	35.715615	65.676028	-11.549151	61.531612	-5.504689	1.351058	44.800517	22.684800	78.034358	41.540218
4	-262.935166	-106.227740	-27.991550	-5.783761	7.755305	30.451924	79.495711	-25.954843	48.446620	2.812927	21.386844	44.666454	-6.020085	51.143096	33.886077
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
495	-221.980369	-91.835973	-24.365038	-9.114581	15.297867	26.887518	91.097365	-33.412618	50.384399	-2.724579	21.994533	38.605246	-0.705895	53.434874	36.995532
496	-189.206062	-81.097693	-22.366217	-11.625391	4.284231	28.115651	91.961946	-42.030972	52.772717	-3.308277	23.255891	37.898573	-0.219152	54.446121	42.245011
497	-294.395124	-108.410106	-28.125458	-47.217965	9.914564	53.583306	79.543128	-4.270872	83.412555	-9.274723	-15.641246	50.869674	45.521312	120.303705	57.943448
498	-249.970066	-96.336212	-26.286065	-18.951135	5.703226	30.369091	67.747374	-12.408518	58.785526	-6.275651	3.757532	39.211886	21.729272	71.541331	40.638277
499	-233.720177	-90.301027	-27.227860	-29.649053	3.243316	29.828477	79.947034	-17.336223	64.194158	-7.973780	4.335782	39.832601	28.638381	86.696391	47.172322

500 rows × 15 columns

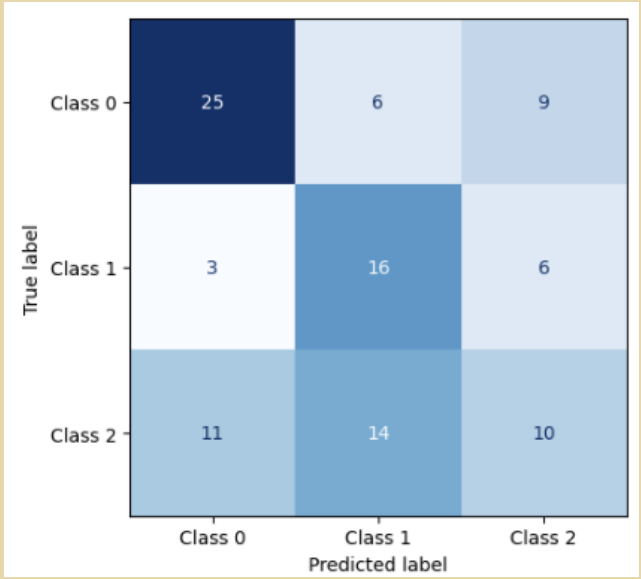
- PCA를 통한 적합한 PC 개수 추출 = 15개

IV. 실험 및 평가

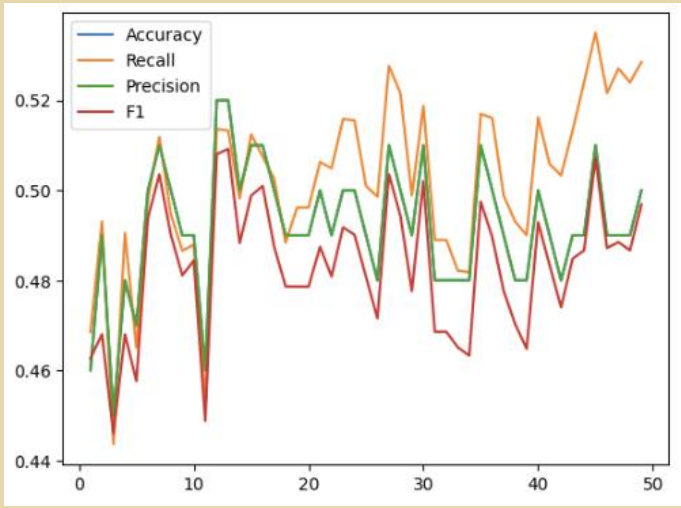
K-NN Classification + PCA

```
# Display confusion matrix with appropriate class labels
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_labels)
disp.plot(cmap='Blues', colorbar=False)
```

```
Accuracy: 0.510
Recall Score: 0.510
Precision Score: 0.508
F1 Score: 0.501
```



Resulting Confusion Matrix



n_neighbor decision

- Accuracy = 0.510
- F1 Score = 0.501

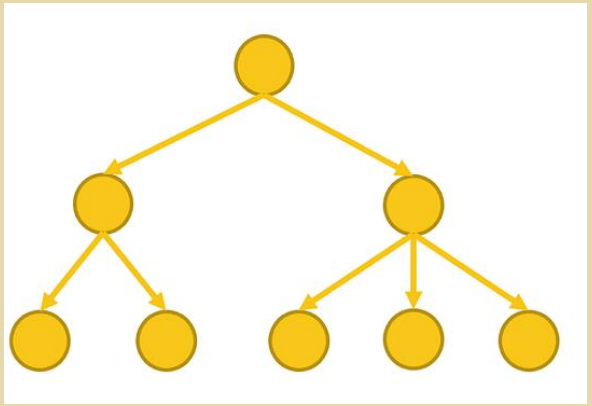
→ 전반적인 성능 하락

IV. 실험 및 평가

Tree-based Classifiers

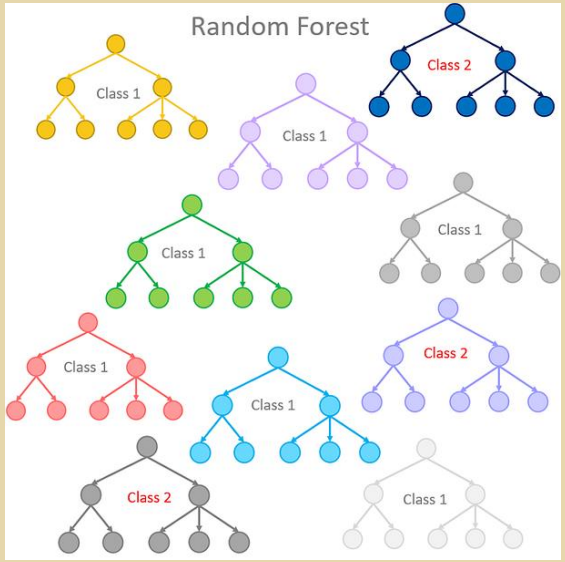
- 단일 의사결정 나무 → 의사결정 나무의 집합 → 더욱 random한 Random Forest
- 모델 별로 출력에 영향을 주는 특성의 중요도를 다르게 평가

Decision Tree

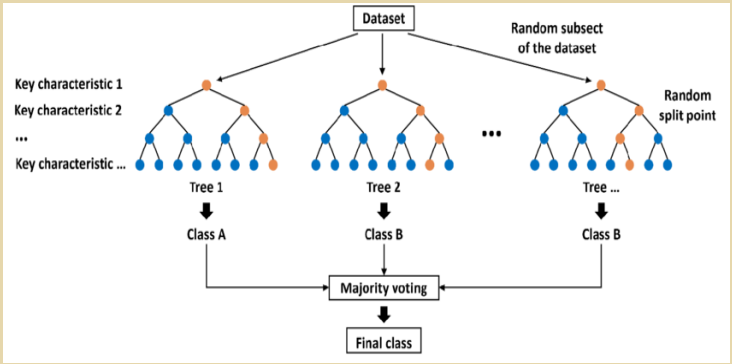


https://towardsdatascience.com/from-a-single-decision-tree-to-a-random-forest-b9523be65147

Random Forest



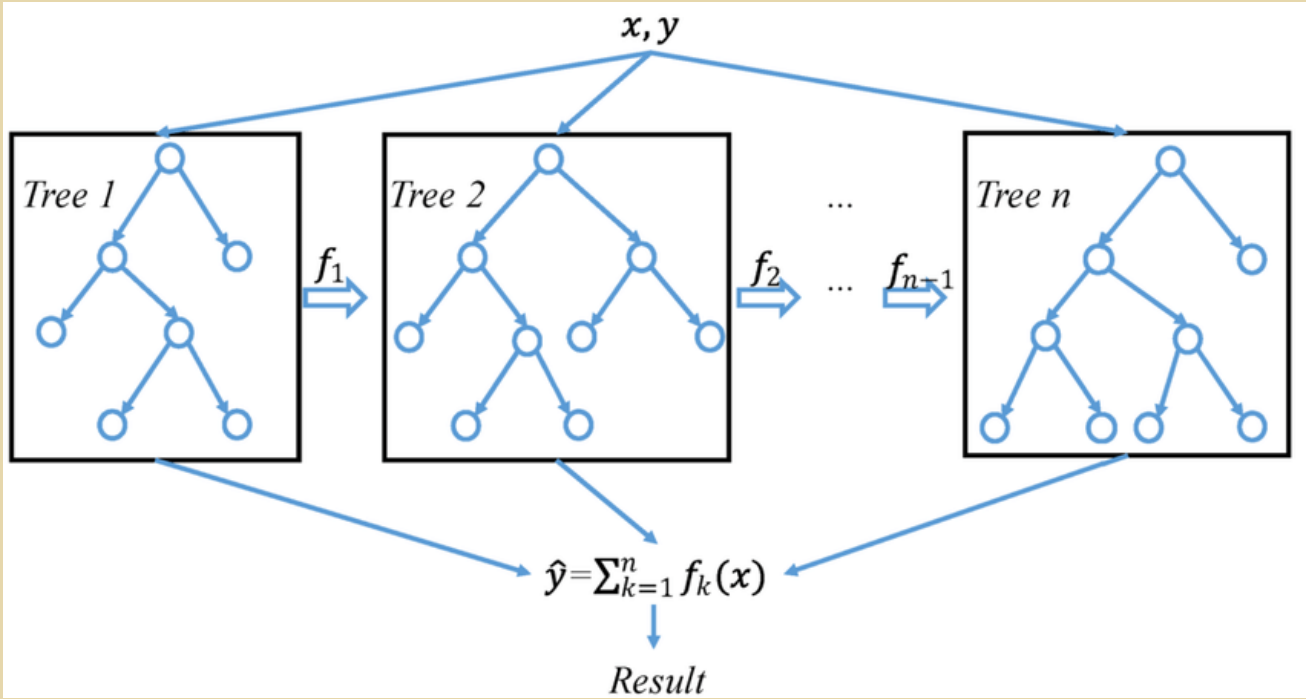
Extra Forest



https://www.researchgate.net/figure/Structure-of-Extra-Trees-Kapoor-2020-Extra-Trees-constructs-the-set-of-decision-trees_fig1_364771403

IV. 실험 및 평가

XGBoost



https://www.researchgate.net/figure/A-general-architecture-of-XGBoost_fig3_335483097

- Random Forest와 비슷하게 Decision Tree의 Ensemble
- 이전 Tree의 예측 오차를 기반으로 새로운 Tree 훈련
- 이전 Gradient Boosting Model 보다 좋은 성능 + 빠른 속도

IV. 실험 및 평가

Decision Tree & Tree Ensemble Models

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size = 0.2, random_state = 2023)
```

```
X_test = X_test.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)
```

```
x_shuffled = sklearn.utils.shuffle(X_train, random_state=312)
y_shuffled = sklearn.utils.shuffle(y_train, random_state=312)

smote = SMOTE(random_state=312)
X_train, y_train = smote.fit_resample(x_shuffled, y_shuffled)
```

```
r = 2023
classifiers = {
    'DTC': DecisionTreeClassifier(random_state=r),
    'RFC': RandomForestClassifier(random_state=r),
    'XGB': XGBClassifier(random_state=r),
    'ETC': ExtraTreesClassifier(random_state=r)
}

for name in classifiers:
    score = cross_val_score(classifiers[name], X_train, y_train, cv=5, scoring='accuracy').mean()
    print(name, score.round(4))
```

```
DTC 0.6205
RFC 0.7107
XGB 0.807
ETC 0.6579
```

- Train : Test = 8 : 2 분리
- Cross-Validation = 5 수행
- Train Data 기준 XGB가 80.7%의 정확도로 최고의 성능을 보여줌
- XGB를 최적의 모델로 선정, grid search를 통한 hyperparameter tuning

IV. 실험 및 평가

XGBoost – hyperparameter grid search

```
param_grid = [{'xgbclassifier__max_depth': [None, 3, 6, 9, 12],
               'xgbclassifier__min_child_weight': [1, 3, 5, 7, 10],
               'xgbclassifier__n_estimators': [50, 100, 150, 200]
              }]

gs = GridSearchCV(estimator = pipe,
                  param_grid = param_grid,
                  scoring = 'f1_macro',
                  cv=10,
                  n_jobs= -1, error_score='raise', verbose=3)

gs = gs.fit(X_train, y_train)

print(gs.best_score_)
print(gs.best_params_)
```

```
[CV 8/10] END xgbclassifier__max_depth=9, xgbclassifier__min_child_weight=5, xgbclassifier__n_estimators=100;, score=0.819 total time=   0.5s
[CV 4/10] END xgbclassifier__max_depth=9, xgbclassifier__min_child_weight=5, xgbclassifier__n_estimators=150;, score=0.761 total time=   0.6s
[CV 10/10] END xgbclassifier__max_depth=9, xgbclassifier__min_child_weight=5, xgbclassifier__n_estimators=150;, score=0.820 total time=   0.6s
0.8283695344070399
{'xgbclassifier__max_depth': None, 'xgbclassifier__min_child_weight': 3, 'xgbclassifier__n_estimators': 150}
```

- XGBoost에서 가지는 hyperparameter max_depth, min_child_weight, n_estimators를 왼쪽과 같이 다양하게 설정하고, grid search를 실행해 macro f1 score (f1-score의 산술평균)가 가장 높은 hyperparameter 선정

→ Cross-Validation = 10으로 설정, max_depth = None, min_child_weight = 3, n_estimators = 150으로 설정했을 때 train 의 macro f1 score가 0.8283으로 산출

IV. 실험 및 평가

XGBoost – hyperparameter grid search

```
[59]: model = XGBClassifier(max_depth=None, min_child_weight=3, n_estimators=150, random_state=0)
      score = cross_val_score(model, X_train, y_train, cv=5, scoring='f1_macro').mean()
      print(score.round(4))

      0.8296

[60]: model.fit(X_train, y_train)
      y_pred = model.predict(X_test)

[61]: model.save_model("model.json")

[62]: print('정확도 accuracy: %.3f' % accuracy_score(y_test, y_pred))
      print('정밀도 precision: %.3f' % precision_score(y_true=y_test, y_pred=y_pred, average='macro'))
      print('재현율 recall: %.3f' % recall_score(y_true=y_test, y_pred=y_pred, average='macro'))
      print('F1-score: %.3f' % f1_score(y_true=y_test, y_pred=y_pred, average='macro'))

      정확도 accuracy: 0.740
      정밀도 precision: 0.737
      재현율 recall: 0.718
      F1-score: 0.725
```

- 정해진 hyperparameter를 바탕으로 Cross-validation = 5로 축소 (연산 속도 개선), 최종 모델 결정
- 구한 최적 모델을 'model.json'으로 별도 extract
- 최적 모델을 Test Data에 적용

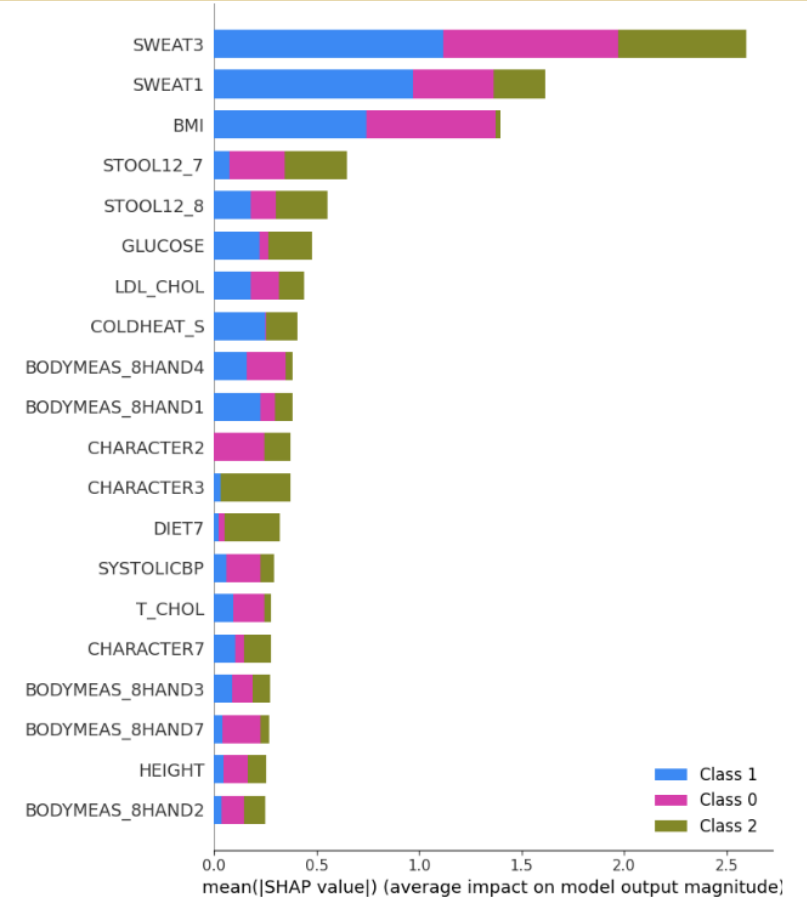
→ Test Data 성능

Accuracy = 0.740 Recall = 0.718

Precision = 0.737 F1 Score : 0.725

IV. 실험 및 평가

XGBoost – hyperparameter grid search



- SHAP value 기준 사상체질 분류에 가장 큰 영향을 주는 요소는 SWEAT3, SWEAT1,BMI임
- 실제로 한의학에서 규정하는 사상체질 분류 기준 중 체중 및 체형이 많은 영향을 준다는 것을 알 수 있음
- 더욱 많은 데이터를 학습시키거나, 주관적 요소가 들어갈 수 있는 '정도' 표기 데이터가 아닌 숫자 데이터가 주어진다면 실질적으로 영향을 주는 요소를 정확하게 판별할 수 있음

V. 활용 및 기대 효과

진단 '대체' 가 아닌 '보조' 의 역할 수행

- 데이터만을 바탕으로 체질 진단을 하는 것은 오히려 섣부를 수 있음 → 특히 주어진 데이터가 정량적인 수치 데이터가 아닌, '정도'를 나타내기에 주관적 요소가 포함될 가능성 다분
- 한의사의 체질 진단을 돕는 보조 도구로서의 역할을 수행해야 할 것
- 입력 변수들의 '유의미함'을 판단하는 데에는 입력변수들의 수치적 영향력 뿐만이 아닌, 한의학이 정의한 체질 분류 방법론이 동시에 적용되어야 함 → 전문가의 의견 + 데이터 기반 근거 형성

한의학 대중화

- 간단한 문답으로 구성된 앱 등을 통해 사람들이 손쉽게 체질 자가진단을 할 수 있고, 이를 기반으로 체질별 생활습관 · 식습관 추천 가능
- 한의사와의 상담 및 전문가 의견이 필수적임을 명시해야 할 필요

감사합니다!