고혈압 분석 모델

2021-08-31

컴퓨터과학과 황승현

목차

개요

- 식품영양학과 김혜림 박사님
- 사람의 나이, 영양, 식이 등 생활 패턴과 특정 질환의 상관 관계 조사
  - 식이 패턴과 고혈압의 상관 관계를 집중 연구
- 고혈압 모델 제작
  - 새로운 변수(사람)의 고혈압 유병 여부 예측.
  - 현재는 정상혈압이지만, 이후 고혈압에 걸릴지 예측

고혈압 분석 모델 소개

생활 패턴

독립 변수 [X]

- 검사자 개인정보
  - 성별, 연령, 직업, 교육 수준 등
- 생활습관
  - 음주, 흡연 등
- 생활패턴
  - 신체 활동 시간, 수면 시간, 식사 횟수 등
- 영양소 섭취량
- 식이 패턴
  - P1: 육류, 어패류, 과일류
  - P2: 면류/떡국류, 서류, 빵류, 두류/난류
  - P3: 밥류, 김치류, 채소류
  - P4: 식사대용, 음료, 과자/사탕

고혈압

종속 변수 [y]

- 고혈압 여부 판단 기준에 맞추어 새로운 변수 생성
  - **HYPERTENSION**
- 3가지 경우 중 1개 이상을 만족할 때
  - 누운 자세 – 2회 **sys** 측정 평균 140이상일 경우
  - 누운 자세 – 2회 **dia** 측정평균 90이상일 경우
  - 혈압약 현재 지속여부 '예'일 경우

코드, 라이브러리 설명

지금까지 한 일

- 고혈압 변수 가공
- 데이터 분리 및 전처리
- 결측값 대치
  - 기반 데이터 변수 수정
  - KNNImputer, SimpleImputer
- 변수 스케일링
  - StandardScaler, MinMaxScaler, QuantileTransformer
- 모델 제작 및 하이퍼파라미터 튜닝
  - Train, test 분리
  - Keras Tuner
  - Hyperband

지금까지 한 일

- Decision tree 제작
  - DecisionTreeClassifier
  - Decision tree로 변수 중요도 추출
- 다양한 모델 설계
  - 변수 솎아내고 모델 설계
  - 식이패턴을 이용한 모델 설계
  - 결측값을 모두 대치한 모델 설계

기반데이터 변수 수정

- 기반데이터 변수의 특징
  - 66666 = 조사 안 함
  - 77777 = 해당 없음
  - 99999 = 미상/무응답/미측정
  - 그 외, 비어 있는 값.
- 변수의 특성에 맞게 0, NA 등으로 수정

데이터 분리

```
# dataset에 있는 변수 분리
# 범주형, 연속형 등으로 분리하여 raw_var 형태로 저장하여 각각 관리한다.

# 종속변수, 고혈압
HYPERTENSION = dataset['HYPERTENSION']

# 범주형(binary, 0 or 1)
raw_binary = dataset.reindex(columns=['AS1_SEX', 'AS0_TIED', 'AS0_SLPAMSF', 'AS1_STRPHYSJ'])
col_b = raw_binary.columns

#범주형(계층 없음, without hierarchy)
raw_categoryH0 = dataset.reindex(columns=['AS1_JOBB', 'AS1_INSUR'])
col_H0 = raw_categoryH0.columns

# 범주형(계층 있음, with hierarchy)
raw_categoryH1 = dataset.reindex(columns=['AS1_EDUA', 'AS1_INCOME', 'AS1_DRINK', 'AS1_DRDUA',
                                          'AS1_SMOKEA', 'AS1_PHYSTB', 'AS1_PHYSIT', 'AS1_PHYACTL',
                                          'AS1_PHYACTM', 'AS1_PHYACTH', 'AS1_HEALTH'
                                          ])
col_H1 = raw_categoryH1.columns

# 연속형 변수
raw_ctn = dataset.reindex(columns=['AS1_AGE', 'AS1_HVSMAM', 'AS1_HVSMDU', 'AS1_TOTALC',
                                   'AS1_SLPAMTM', 'AS1_RGMEALFQA', 'AS1_HEIGHT', 'AS1_WEIGHT',
                                   'AS1_B01', 'AS1_B02', 'AS1_B03', 'AS1_B04', 'AS1_B05',
                                   'AS1_B06', 'AS1_B07', 'AS1_B08', 'AS1_B09', 'AS1_B10',
                                   'AS1_B11', 'AS1_B12', 'AS1_B13', 'AS1_B14', 'AS1_B15',
                                   'AS1_B16', 'AS1_B17', 'AS1_B18', 'AS1_B19', 'AS1_B20',
                                   'AS1_B21', 'AS1_B23', 'AS1_B24', 'P1', 'P2', 'P3', 'P4'
                                   ])
col_c = raw_ctn.columns
```

```
print(raw_binary.info(), raw_categoryH0.info(), raw_categoryH1.info(), raw_ctn.info())
print(col_b, col_H0, col_H1, col_c)
```

Imputer

- scikit-learn impute
- KNNImputer
  - 최근접 이웃의 평균값을 사용하여 대치
- SimpleImputer
  - 결측값을 평균, 중앙값, 최빈값, 상수 등으로 대치

One-hot-encoding

```
imp_kNN = KNNImputer(n_neighbors=5)
imp_mean = SimpleImputer(strategy='mean')
imp_mostFreq = SimpleImputer(strategy='most_frequent')

# 최빈값으로 대치
binary = pd.DataFrame(imp_mostFreq.fit_transform(raw_binary),
                      index=idx_dataset, columns=col_b).astype('float') # float형으로 변경.

categoryH0 = pd.DataFrame(imp_mostFreq.fit_transform(raw_categoryH0),
                          index=idx_dataset, columns=col_H0)
categoryH0 = pd.get_dummies(categoryH0, columns=col_H0).astype('float') # one-hot-encoding

categoryH1 = pd.DataFrame(imp_mostFreq.fit_transform(raw_categoryH1),
                          index=idx_dataset, columns=col_H1)

# kNN, 평균 둘 중 하나만 사용할 예정
ctn = pd.DataFrame(imp_kNN.fit_transform(raw_ctn), index=idx_dataset, columns=col_c)
```

- **pd.get_dummies()**
- 변수를 벡터로 표현
- **One-Hot-Encoding**을 하는 이유
  - 정수 값으로 두면 각 수가 연관성이 있는 것이라고 착각하기 때문

```
1 raw_categoryH0
```

RID	AS1_JOBB	AS1_INSUR
EPI20_026_2_000001	1.0	3.0
EPI20_026_2_000002	3.0	2.0
EPI20_026_2_000003	7.0	2.0
EPI20_026_2_000004	8.0	2.0
EPI20_026_2_000005	3.0	2.0
...	...	...
EPI20_026_2_010026	4.0	2.0
EPI20_026_2_010027	1.0	2.0
EPI20_026_2_010028	4.0	2.0
EPI20_026_2_010029	3.0	2.0
EPI20_026_2_010030	3.0	2.0

9704 rows × 2 columns

```
1 categoryH0
```

	AS1_JOBB_0.0	AS1_JOBB_1.0	AS1_JOBB_2.0	AS1_JOBB_3.0
EPI20_026_2_000001	0.0	1.0	0.0	0.0
EPI20_026_2_000002	0.0	0.0	0.0	1.0
EPI20_026_2_000003	0.0	0.0	0.0	0.0
EPI20_026_2_000004	0.0	0.0	0.0	0.0
EPI20_026_2_000005	0.0	0.0	0.0	1.0
...	...	...	...	...
EPI20_026_2_010026	0.0	0.0	0.0	0.0
EPI20_026_2_010027	0.0	1.0	0.0	0.0
EPI20_026_2_010028	0.0	0.0	0.0	0.0
EPI20_026_2_010029	0.0	0.0	0.0	1.0
EPI20_026_2_010030	0.0	0.0	0.0	1.0

9704 rows × 12 columns

One-Hot Encoding

데이터 스케일링

- 데이터 스케일링이란?
  - 데이터를 일정 범위 안으로
  - 모델의 학습효율 향상
- **Scikit-learn preprocessing**
- **StandardScaler**
  - 평균과 분산을 이용하여 변수 스케일링
  - 표준화
- **MinMaxScaler**
  - 최솟값과 최댓값을 이용하여 변수를 스케일링
  - 정규화
- **QuantileTransformer**
  - 4분위수를 이용하여 변수를 스케일링
  - 이상치에 잘 대처

Keras Tuner

- 하이퍼 튜닝을 도와주는 라이브러리
- 하이퍼 튜닝
  - **hyperparameter tuning**
  - 모델(**hyper model**)에서 가장 좋은 하이퍼파라미터를 찾는 것
- **Hyperband** 튜너를 인스턴스화 하여 사용

```python
class RegressionHyperModel(HyperModel):
  def __init__(self, input_shape):
    self.input_shape = input_shape

def model_builder(hp):
  model = Sequential()
  hp_units = hp.Int('units', min_value = 4, max_value = 64, step = 2)
  hp_dropout = hp.Float('dropout', min_value=0.0, max_value=0.5, default=0.05, step=0.05)

  model.add(Dense(units = hp_units,
                  activation='relu',
                  input_shape=input_shape)) # input_shape = 62

  model.add(Dropout(hp_dropout))

  model.add(Dense(units = hp_units,
                  activation='relu'))

  model.add(Dropout(hp_dropout))

  model.add(Dense(units = hp_units,
                  activation='relu'))

  model.add(Dropout(hp_dropout))

  model.add(Dense(units = hp_units,
                  activation='relu'))

  model.add(Dropout(hp_dropout))

  model.add(Dense(units = hp_units,
                  activation='relu'))

  model.add(Dropout(hp_dropout))

  model.add(Dense(1, activation='sigmoid')) # 출력층

  # Tune the learning rate for the optimizer S
  hp_learning_rate = hp.Choice('learning_rate', values = [1e-3]) #0.001

  model.compile(optimizer = keras.optimizers.Adam(learning_rate = hp_learning_rate),
                loss="binary_crossentropy", # 손실함수: binary_crossentropy
                metrics = ['accuracy']) # 평가지표

  # model.compile(optimizer='rmsprop',
  #               loss='mse', metrics=['mse']) #손실함수: MSE(mean squared error)

  return model
```

Hyper Model

- 고혈압 모델 설계
- **model**
  - **Sequential()**
- **hp_units**
  - **hidden layer의 노드 수**
- **hp_dropout**
  - **dropout 확률**
- **model.add**
  - **layer 추가**

```
tuner = kt.Hyperband(model_builder,
                     objective = 'val_accuracy',
                     max_epochs = 64,
                     hyperband_iterations = 100,
                     directory = '/content/drive/MyDrive/Colab Notebooks',
                     project_name = 'HyperTension_sh22h10')
```

```
11 tuner.search(X_train, y_train,
12                 epochs = 10,
13                 validation_split=0.2,
14                 callbacks = [ClearTrainingOutput()])
15
```

```
Trial 263 Complete [00h 00m 11s]
val_accuracy: 0.7636831998825073

Best val_accuracy So Far: 0.7707662582397461
Total elapsed time: 00h 19m 16s

Search: Running Trial #264

Hyperparameter    |Value             |Best Value So Far
units             |62                |22
dropout           |0.45              |0
learning_rate     |0.001             |0.001
tuner/epochs      |2                 |10
tuner/initial_e...|0                 |0
tuner/bracket     |2                 |0
tuner/round       |0                 |0

Epoch 1/2
195/195 [==============================] - 1s 4ms/step - loss: 0.5715 - accuracy: 0.7567 - val_loss: 0.5605 - val_accuracy: 0.7624
```

Hyperband 튜너

- 최고 성능을 보이는 절반만 다음 단계로 넘김
- **model_builder**
  - **HyperModel**, 고혈압 모델 인스턴스
- **Objective:** 최적화 방향
  - **'val_accuracy'**로 모델의 성능 평가
  - 검증 정확도
- **Max_epochs**
  - 모델을 학습시키는 최대 **Epoch** 수
- **hyperband_iterations**
  - **Hyperband** 알고리즘을 반복할 횟수

Decision Tree

- scikit-learn.tree DecisionTreeClassifier
  - DecisionTree(결정 트리)를 만듦
- Feature_importances_
  - 변수 중요도
  - Decision Tree 를 쓰는 이유!
  - 변수 중요도로 고혈압에 영향을 주는 변수 판단!

```
[21]  1 from sklearn.tree import DecisionTreeClassifier
      2 from sklearn.tree import export_graphviz
      3 import graphviz
      4
      5 ht_tree = DecisionTreeClassifier(max_depth=5)
      6 ht_tree.fit(X_test, y_test)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=5, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
[22]  1 export_graphviz(ht_tree, out_file="tree.dot",
      2                 class_names='FT',
      3                 feature_names=X.columns,
      4                 impurity=True, filled=True)
```

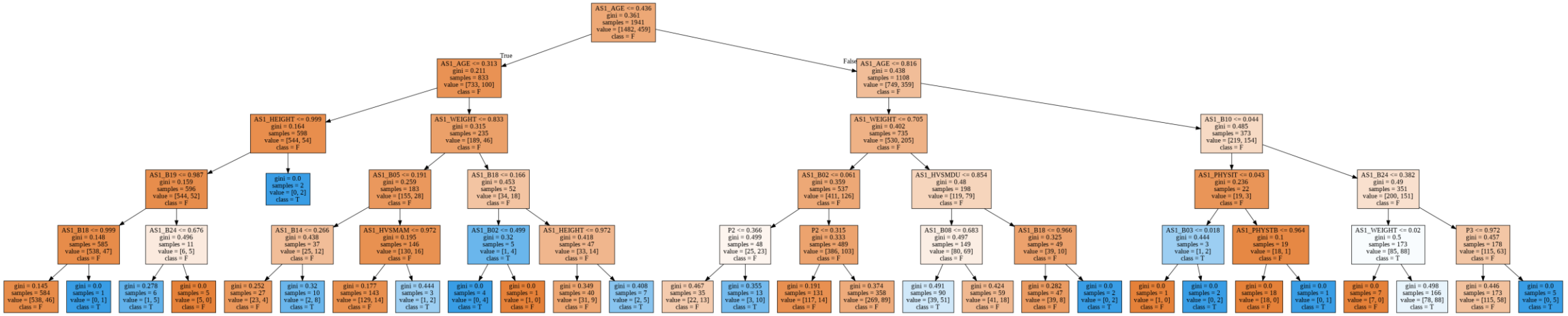
```
[17]  1 !ls
```

```
drive  sample_data  tree.dot
```

```
[23]  1 with open("tree.dot") as f:
      2     dot_graph = f.read()
      3
      4 graphviz.Source(dot_graph, filename='tree.png',format='png')
```

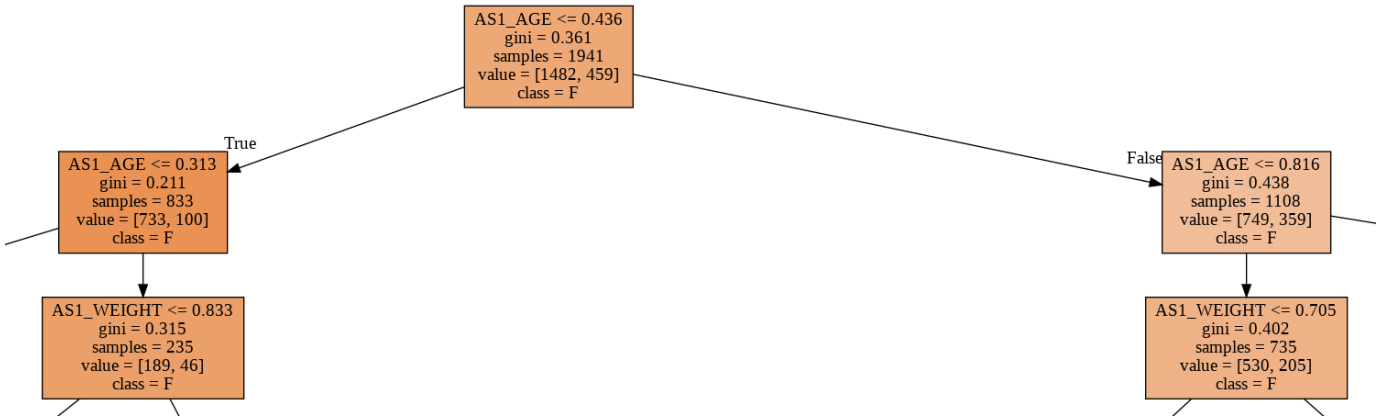
Decision Tree

- **fit()**
  - 의사결정트리 생성
- **Export_graphviz()**
  - 트리를 .dot 파일로 내보냄
  - 트리를 .png 파일로 내보냄

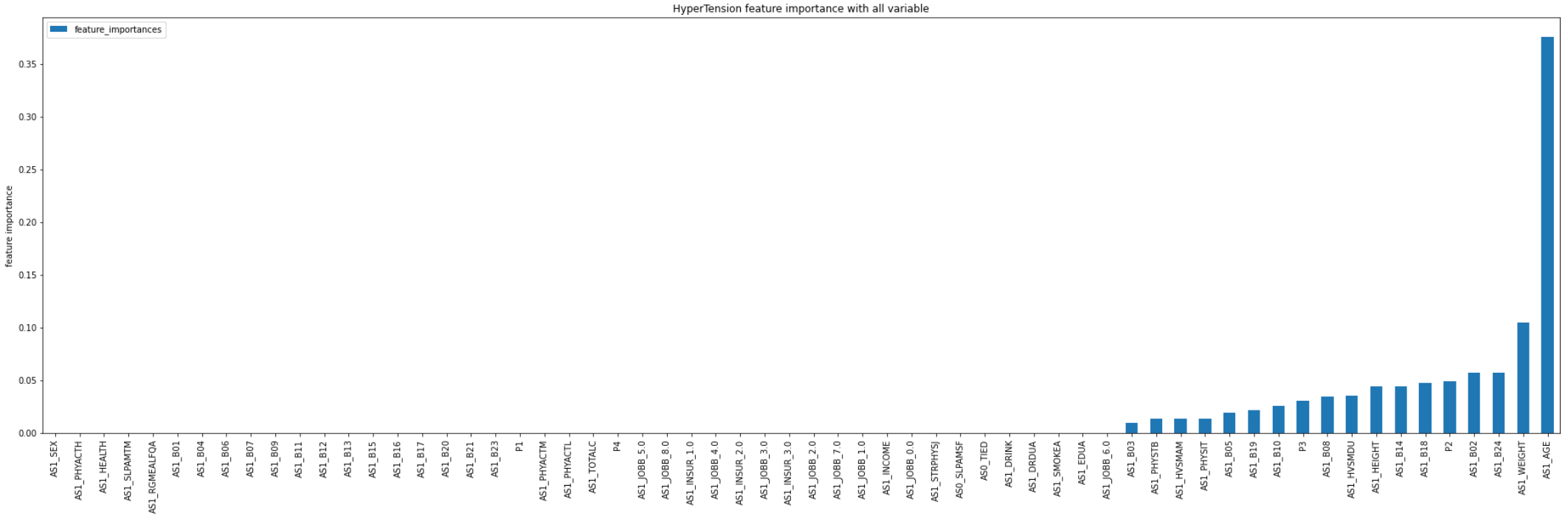


Decision Tree (max_depth = 5)

Decision Tree의 해석

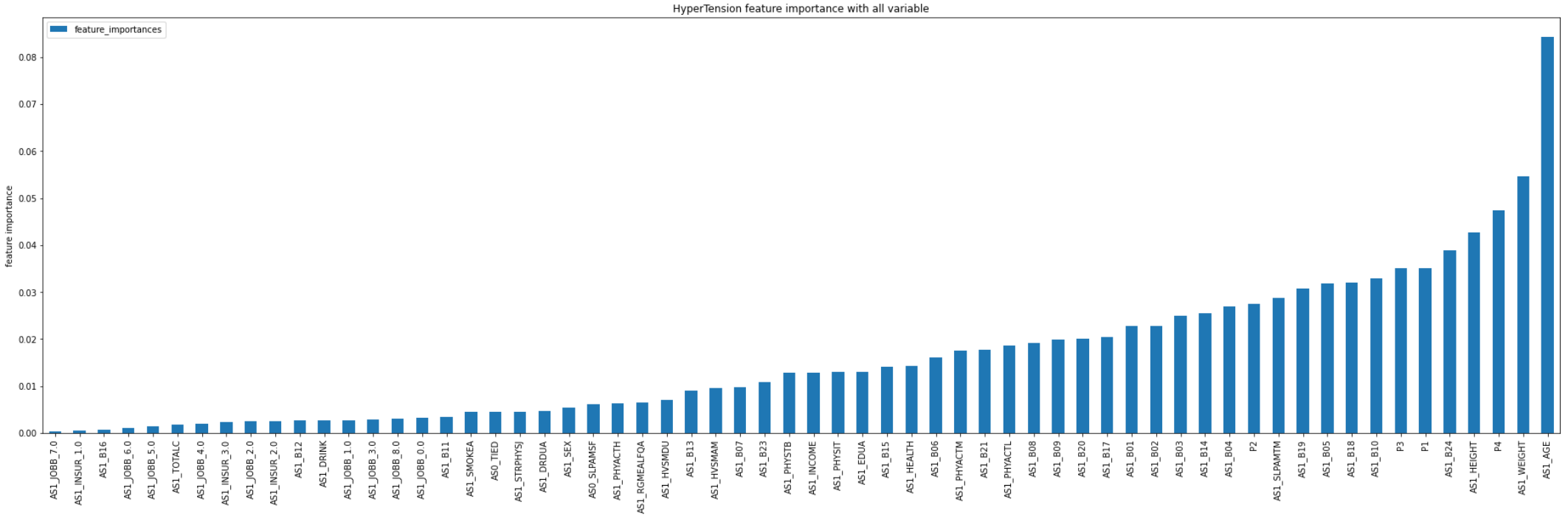


- AS1_AGE <= 0.436
  - 분할조건
  - 노드를 쪼개는 기준
- Gini = 0.361
  - 노드의 지니 불순도
- Samples = 361
  - 노드에 있는 정보의 개수
- Value = [1482, 459]
  - Decision Tree로 분리한 샘플의 수



Feature importance

- max_depth = 5



Feature importance

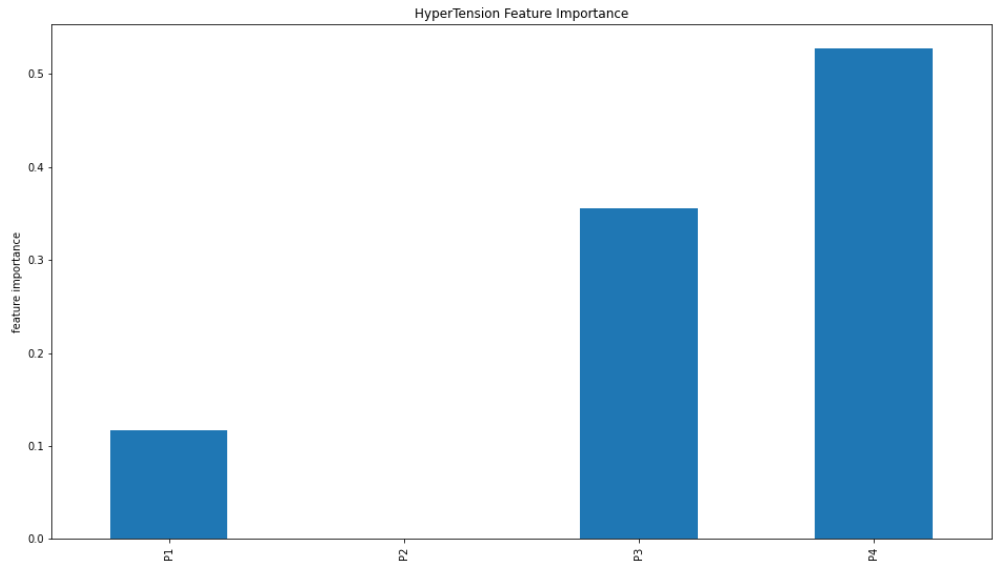
- **max_depth** 없음.
- 파일 크기 약 **7Mb**. 부록 참고 바람

변수를 솎아낸 모델

```
1 X_cut10 = X[importances_rank[10:]] # 변수 중요도 하위 10개 제거
2 X_cut20 = X[importances_rank[20:]] # 변수 중요도 하위 20개 제거
3 X_cut30 = X[importances_rank[30:]]
4 X_cut40 = X[importances_rank[40:]]
5 X_cut50 = X[importances_rank[50:]]
6
```

- 변수 중요도 그래프 참고
- 변수 중요도 순위에 따라 **X**를 나누어 각각의 모델 제작
- 극적인 정확도 상승은 없음.

식이패턴 모델



- 식이패턴만을 x로 둔 모델
- 극적인 정확도 상승은 없음.

결측값을 모두 대치한 모델

- 이전에는 결측값이 있는 행을 모두 제거하여 모델 제작
  - 9704개의 데이터 중 1000개만 사용
- Imputer로 모든 변수의 결측값을 대치함
  - 9704개의 데이터를 온전히 사용함

성과, 추후 계획

성과

- KOGES 데이터셋 규격화 및 스케일링
  - 모델을 다양하게 변형할 수 있음
  - 후속 연구 및 개발에 쉽게 적용할 수 있음
- 고혈압 모델 정확도 78%까지 상승
  - 목표: 90% 이상
- 머신러닝 지식 습득
  - 모델 설계와 학습
  - NumPy, Pandas, Scikit-learn, Keras, Keras_Tuner
  - 다른 머신러닝 과제를 할 때 도움이 될 것이다

추후 계획

- 모델 정확도 보는 법 정리할 것
  - evaluate()
- 변수의 상관관계 결정방법 정리할 것
  - Feature impotance
- 다양한 Decision Tree 제작할 것
  - Decision Tree의 나타난 정보 분석

- 모델 정확도 향상을 위해 이 한 몸 불사를 것
- 영광스러운 논문의 완성을 위해 결사보위의 태세로 덤빌 것
- 일이 다 끝나면 시원한 맥주 한 잔 할 것

감사합니다~