Multivariate Data Analysis Assignment #3

Decision Tree & Neural Network

산업경영공학부 2020170831 민찬홍

[Q1] 데이터셋 선정 및 선정 이유

데이터링크: https://www.kaggle.com/datasets/tawfikelmetwally/employee-dataset

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4653 entries, 0 to 4652
Data columns (total 9 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Education                4653 non-null   object
 1   JoiningYear              4653 non-null   int64
 2   City                     4653 non-null   object
 3   PaymentTier              4653 non-null   int64
 4   Age                      4653 non-null   int64
 5   Gender                   4653 non-null   object
 6   EverBenched              4653 non-null   object
 7   ExperienceInCurrentDomain  4653 non-null   int64
 8   LeaveOrNot               4653 non-null   int64
dtypes: int64(5), object(4)
memory usage: 327.3+ KB
None
```

본 데이터셋은 'Employee dataset'으로 교육 수준, 입사 년도 등의 고용과 관련된 요소와 직원의 회사를 떠나는지 여부를 담고 있는 dataset이다. 종속변수인 'LeaveOrNot'과 독립변수 8개를 합한 총 9개의 column으로 구성되어 있으며 데이터의 개수는 4653이다. 특별히 종속변수인 'LeaveOrNot'의 경우 Leave(1), Not Leave(0)의 2개의 Class로 이루어져 있기에 binary classification 문제이다.

본 데이터셋은 직장을 떠나는지 여부를 이와 관련된 요인들로 예측할 수 있기에 예측 정확도가 중요하다. 만약 한 직원이 곧 떠난다는 것을 예측할 수 있다면 미리 부서를 옮기거나 프로젝트에서 배제시키는 등 조치를 취할 수 있을 것이다. 하지만, 기업 입장에서 직원이 떠나지 않도록 미리 예방할 수 있다면 이러한 조치를 취할 필요가 없을 것이다. 따라서 장기적인 측면에서 보았을 때 직장을 떠나는지 여부에 영향을 미치는 관련 요소에 대한 해석은 매우 중요하다고 할 수 있다. 이 해석을 통해 기업 입장에서는 직원들의 이직, 퇴사율을 낮출 수 있을 것이라 생각하고, 직원들은 기업 만족도가 올라가는 선순환 구조를 만들 수 있을 것이라 생각하였다.

학습: 검증: 테스트 데이터셋의 분배 비율로 7:1.5:1.5의 비율을 사용하였다. 일반적으로 학습: 검증: 테스트 데이터셋의 분배 비율은 60~80%, 10~20%, 10~20%이다. 만약, 학습데이터의 비중을 80%로 가져간다면 그 만큼 많은 데이터를 학습할 수 있기에 예측 성능은 좋아질 것으로 기대되나 최적의 하이퍼파라미터 설정 및 테스트 데이터를 예측하는 데 부족함이 있을 것이다. 따라서

테스트 데이터 예측 결과물에 대한 해석이 부족할 것이고, 본 데이터셋은 예측 성능보다 예측 결과물에 대한 해석이 더 중시되어야 한다고 생각하기에 학습데이터셋의 비중을 70%로 가져갔다. 이 때, 전체 데이터의 개수가 4653개로 충분하기에 학습데이터셋의 비중을 70%로 가져가도 충분한 학습데이터를 확보할 수 있다고 보았다.

[Q2] Full tree, Post-pruning tree 분류 성능 평가 및 비교

Decision Tree로 분류하기에 앞서 결측치 제거, 명목형 변수 원핫인코딩, scaling 작업을 거쳐 데이터를 가공하였다.

```
Education                  0
JoiningYear                0
PaymentTier                0
Age                        0
Gender                     0
EverBenched                0
ExperienceInCurrentDomain  0
LeaveOrNot                 0
City_Bangalore             0
City_New Delhi             0
City_Pune                  0
dtype: int64
```

위 그림에서 알 수 있듯이 결측치는 존재하지 않았기에 결측치 제거 작업을 수행하지 않았다.

이후 명목형 변수 데이터 가공을 진행하였다. 독립 변수들 중 총 4개의 변수(Education, City, Gender, Everbenched)가 명목형 변수인 것을 확인할 수 있었다. 이 중 Education의 경우 Bachelors, Masters, PHD 3개의 class로 이루어져 있는데 이 class 간의 서열 관계가 있다고 판단하여 각각 1, 2, 3으로 labeliing encoding을 진행하였다. Gender의 경우 Male을 1, Female을 0으로 Everbenched의 경우 Yes를 1, No를 0으로 처리하였다. City열에 대해서는 원핫인코딩을 진행하였다. 마지막으로 NN 작업을 위해 Scaling 작업을 진행하였다. 이 때 앞서 살펴본 명목형 변수를 제외하고 Scaling을 수행하였다.

	TPR	Precision	TNR	Accuracy	BCR	F1-Measure
Full Tree	0.669604	0.7343	0.883227	0.813754	0.769033	0.700461
Post-Pruning	0.726872	0.850515	0.938429	0.869628	0.825904	0.783848

	AUROC
Full Tree	0.790969
Post-Pruning	0.872106

<Full Tree vs Post-Pruning>

2-1) Full Tree

-양성 클래스를 정확히 예측하는 비율인 TPR이 TNR보다 약 0.21 정도 작고, 차이가 적지 않기에 유의미하다고 판단된다. 이를 해석하면 분류 모델이 Target data의 class의 대한 예측보다 Target data가 아닌 class의 예측을 더 잘한다고 해석할 수 있다.

-양성 클래스로 예측된 것들 중 실제 양성 클래스 비율인 Precision은 약 0.73이다.

-ACC, BCR, F1-Measure의 값은 각각 약 0.81, 0.77, 0.70으로 감소된 형태를 보였다. ACC는 약 81% 수준으로 양성과 음성 case의 구분을 모델이 준수하게 수행한다고 해석할 수 있다. TPR, TNR을 종합적으로 고려한 지표인 BCR 역시 약 77% 수준으로 준수하게 나왔다. TPR, Precision을 종합적으로 고려한 지표인 F1-Measure 역시 약 70% 수준으로 준수하게 나왔다. AUROC의 경우 양성과 음성 case를 구분하는 모델의 능력을 측정하는 척도로, 약 79%인데 이는 합리적인 수준의 판별력이라고 할 수 있다.

2-2) Post-Pruning

-양성 클래스를 정확히 예측하는 비율인 TPR이 TNR보다 약 0.20 정도 작고, 차이가 적지 않기에 유의미하다고 판단된다. 이를 해석하면 분류 모델이 Target data의 class의 대한 예측보다 Target data가 아닌 class의 예측을 더 잘한다고 해석할 수 있다.

-양성 클래스로 예측된 것들 중 실제 양성 클래스 비율인 Precision은 약 0.85이다.

-ACC, BCR, F1-Measure의 값은 각각 약 0.87, 0.83, 0.78으로 감소된 형태를 보였다. ACC는 약 87% 수준으로 양성과 음성 case의 구분을 모델이 준수하게 수행한다고 해석할 수 있다. TPR, TNR을 종합적으로 고려한 지표인 BCR 역시 약 83% 수준으로 준수하게 나왔다. TPR, Precision을 종합적으로 고려한 지표인 F1-Measure 역시 약 78% 수준으로 준수하게 나왔다. AUROC의 경우 양성과 음성 case를 구분하는 모델의 능력을 측정하는 척도로, 약 87% 인데 이는 합리적인 수준의 판별력이라고 할 수 있다.

2-3) Full-tree vs Post-Pruning

모든 성능 지표가 Post-Pruning이 Full-tree보다 높게 나왔고 이를 통해 Post-pruning의 분류 성능이 Full-tree의 분류 성능보다 뛰어남을 확인할 수 있다. Full-tree의 경우 학습 데이터에 대한 과적합 문제로 인해 새로운 데이터에 대한 예측 성능 저하의 위험을 안고 있어 테스트 데이터셋에 대해 상대적으로 낮은 분류 성능을 보인 것으로 해석된다.

그러나 두 모델 모두 TPR, Precision, TNR순으로 분류 성능의 값이 낮았다. 이는 만든 분류 모델이 Target data가 아닌 class의 예측을 Target data인 class에 관한 예측(Precision, TNR)보다 잘한다

고 판단할 수 있다. 이는 종속 변수가 0 class의 경우 3053, 1 class의 경우 1600개로 0 class의 개수가 1class의 개수보다 많아 그 만큼 학습에 더 많이 참여하여 0 class에 대한 예측을 더 잘한다고 해석된다. 또한 ACC, BCR, F1-Measure 순으로 감소하는 형태를 띈다. BCR 지표가 F1-Measure보다 높은 것도 앞선 TPR, Precision, TNR순으로 분류 성능의 값이 낮게 나온 이유와 동일하다고 볼 수 있다. (BCR 지표는 TPR, TNR과 관련된 지표이고, F1-Measure는 TPR, Precision과 관련된 지표이기 때문)

[Q3] Pre-pruning, 하이퍼파라미터 설정 및 최적의 하이퍼파라미터

3-1) Hyperparameter 종류 및 탐색 범위 설정

탐색할 하이퍼파라미터로 아래 표와 같이 총 4가지의 하이퍼파라미터의 값을 변경하며 탐색을 진행하였다.

이름	설명
Criterion	의사결정나무의 split의 quality를 측정하는 기준을 결정
Max_depth	의사결정나무의 maximum depth을 설정
Min_samples_split	Internal node를 split하는데 요구되는 최소 표본의 개수
Min_samples_leaf	Leaf node에서 요구되는 최소한의 표본의 개수를 설정

탐색할 4가지의 하이퍼파라미터의 값은 아래와 같이 설정하고 탐색을 진행하였다.

```
-Criterion: ['gini', 'entropy']

-Max_depth: [None, 5, 10, 20]

-Min_sampels_split: [1, 2, 4]

-Min_samples_leaf:  [2, 5, 10]
```

Criterion의 경우 가장 많이 사용하는 후보인 gini와 entropy를 사용하였다. 각각은 지니 불순도, information gain을 기준으로 분할한다.

Full tree의 깊이를 확인한 결과 23이 나왔다. 따라서 Max_depth의 경우 이보다 작은 값인 None, 5, 10, 20을 사용하였다. 값이 None일 경우 Max_depth의 제한이 없기에 다른 하이퍼파라미터 값에 의해 사전적 가지치기가 진행된다.

Min_samples_split가 Min_samples_leaf의 경우 시행착오법을 통해 가장 좋은 성능을 보여줄 수 있는 후보들을 선택하였다.

3-2) 검증 데이터 AUROC 기준 최적의 하이퍼파라미터 조합

```
from sklearn.model_selection import ParameterGrid

best_auroc=0.0
best_model_pre=None

for params in ParameterGrid(param_grid):
  model=DecisionTreeClassifier(**params, random_state=12345)
  model.fit(x_train, y_train)
  y_val_pred=model.predict(x_val)
  auroc=roc_auc_score(y_val, model.predict_proba(x_val)[:,1])
  if auroc>best_auroc:
    best_auroc=auroc
    best_model_pre=model

y_test_pred=best_model_pre.predict(x_test)
y_test_proba = best_model_pre.predict_proba(x_test)[:, 1]
test_auroc=roc_auc_score(y_test, best_model_pre.predict_proba(x_test)[:,1])

print(best_model_pre.get_params())
print(test_auroc)
```

검증 데이터의 AUROC기준 최적의 하이퍼파라미터 조합은 아래와 같이 나왔다.

`'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 2, 'min_samples_split': 10`

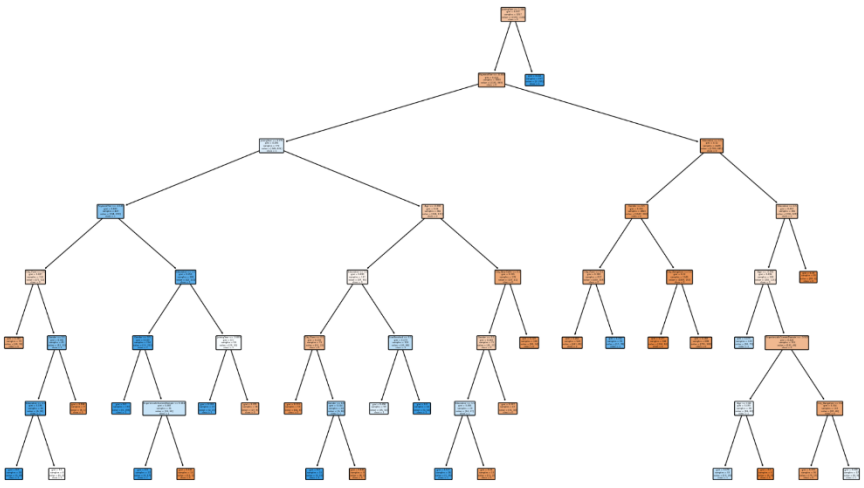
이 때의 AUROC 값은 0.859648이 나왔다.

[Q4] Post-pruning Pre-pruning plot 및 해석, split에 사용된 변수 관찰

(Decision Tree) [Q2]와 [Q3]에서 생성한 Post-pruning 모델과 Pre-pruning 모델의 결과물을 각각 Plotting하고 이에 대한 해석을 수행하시오. 각 Pruning 방식에 따라 Split에 사용된 변수는 어떤 변화가 있는가?

(두 모델의 트리 구조가 한 눈으로 식별하기에 복잡한 관계로 pdf 파일 첨부하겠습니다.)

4-1)Post-pruning

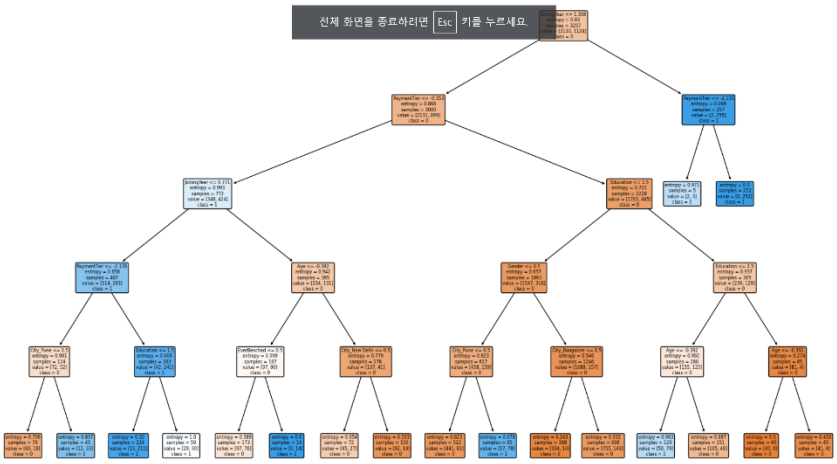


우선적으로, Tree depth는 8임을 알 수 있다. 기존 Full tree의 깊이가 23이었던 것을 감안한다면 확연히 Tree depth가 감소한 것을 확인할 수 있다. 즉, tree의 복잡도가 줄어든 것을 알 수 있다.

Leaf node의 개수는 총 29개이다. 기존 Full tree의 Leaf node의 개수가 670개인 것을 감안한다면 확연히 감소한 것을 확인할 수 있다. 분기에 사용된 변수는 'Education', 'JoiningYear', 'PaymentTier', 'Age', 'Gender', 'EverBenched', 'ExperienceInCurrentDomain', 'City_Bangalore', 'City_New Delhi', 'City_Pune'로 총 10개가 나왔다. 모든 설명 변수가 분기에 사용되었다고 할 수 있고, 분기에 사용된 변수의 개수는 감소하지 않았다.

종합적으로 보면 Full tree와 비교하여 Tree depth와 Leaf 노드의 개수가 감소하긴 하였으나 분기에 사용된 변수의 개수는 동일하게 나왔다. 그러나 Full tree와 비교하여 Tree depth와 Leaf 노드의 개수가 상당 폭으로 감소한 것을 확인할 수 있기에 Tree의 복잡도는 유의미한 수준으로 감소하였고, 그에 따른 설명력은 유의미한 수준으로 증가하였다고 해석할 수 있다.

4-2)Pre-pruning



우선적으로, Tree depth는 6임을 알 수 있다. 기존 Full tree의 깊이가 23이었던 것을 감안한다면 확연히 Tree depth가 감소한 것을 확인할 수 있다. 또한 Leaf node의 개수는 총 18개이다. 기존 Full tree의 Leaf node의 개수가 670개인 것을 감안한다면 확연히 감소한 것을 확인할 수 있다. 이를 통해 tree의 복잡도가 줄어든 것을 확인할 수 있다. 분기에 사용된 변수는 'Education', 'JoiningYear', 'PaymentTier', 'Age', 'Gender', 'EverBenched', 'City_Bangalore', 'City_New Delhi', 'City_Pune'로 총 9개가 나왔다. 모든 설명 변수 중 1개를 제외한 나머지 변수가 모두 분기에 사용되었다고 할 수 있다.

종합적으로 보면 Full tree와 비교하여 Tree depth와 Leaf 노드의 개수가 감소하긴 하였으나 분기에 사용된 변수의 개수는 1개만 감소하였다. 그러나 Full tree와 비교하여 Tree depth와 Leaf 노드

의 개수가 상당 폭으로 감소한 것을 확인할 수 있기에 Tree의 복잡도는 유의미한 수준으로 감소하였고, 그에 따른 설명력은 유의미한 수준으로 증가하였다고 해석할 수 있다.

4-3) Post-pruning vs Pre-pruning

두 모델 모두 Full Tree와 비교하였을 때 트리의 깊이, 말단 노드의 수 관점에서 상당한 감소 폭을 보였기에 기존 Full Tree의 복잡도가 큰 폭으로 감소한 것을 확인할 수 있었다. 두 모델을 plot한 그림에서 알 수 있듯이 Pre-pruning이 Post-pruning보다 복잡도가 단순한 것을 한 눈에 확인할 수 있다. 이는 Pre-pruning의 경우가 Post-pruning의 경우가 트리의 깊이, 말단 노드의 수가 적어 트리의 복잡도가 더 낮기 때문이다. 두 모델 모두 분기에 사용된 변수 종류의 개수를 제외한 나머지 측면에서 기존 Full tree보다 복잡도가 유의미한 수준으로 감소하였기에 두 모델 모두 Full tree에 비해 해석이 용이하다고 판단된다.

[Q5] 최적의 결정나무 plot, 세 가지 규칙

5-1)최적의 결정나무

	TPR	Precision	TNR	Accuracy	BCR	F1-Measure	W
Full Tree	0.669604	0.7343	0.883227	0.813754	0.769033	0.700461	
Post-Pruning	0.726872	0.850515	0.938429	0.869628	0.825904	0.783848	
Pre-Pruning	0.621145	0.844311	0.944798	0.839542	0.766066	0.715736	

	AUROC
Full Tree	0.790969
Post-Pruning	0.872106
Pre-Pruning	0.859648

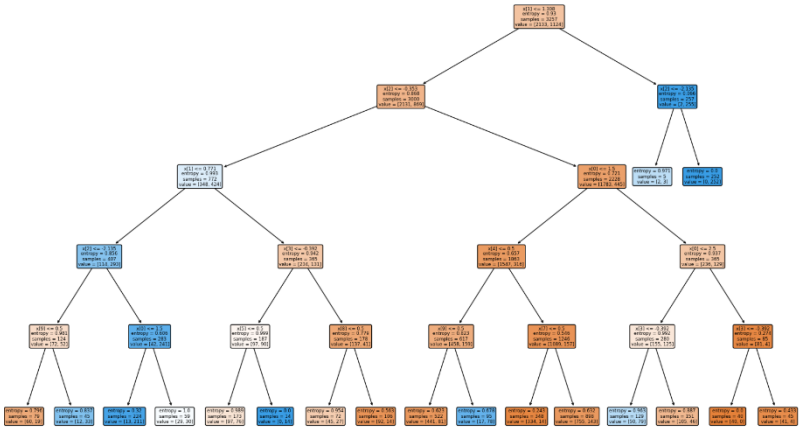
위는 Full Tree, Post-Pruning, Pre-pruning의 분류 성능 지표이다.

Full tree의 경우 깊이가 23, leaf node의 개수가 670개로 모델이 상당히 복잡하고, 학습 데이터에 과적합 우려가 있기에 최적의 모델이라고 볼 수 없다.

앞서 [Q4.]에서 살펴본 Post-Pruning, Pre-Pruning은 각각의 모델들 중에서 최적의 성능을 갖고 있는 모델을 선정하였기에 Post-Pruning과 Pre-Pruning 모델 중 최적의 모델이 곧 전체 최적 모델이라고 할 수 있다.

[Q4.]의 4-3)에서 알 수 있듯이 두 모델 모두 Full tree와 비교하여 복잡도가 유의미한 수준으로 감소함을 확인할 수 있었다. 특별히 Pre-Pruning의 경우 Post-Pruning보다 모델의 복잡도가 더 낮은 것을 확인할 수 있었다. 그러나 모델의 성능 측면에선 Post-Pruning이 Pre-Pruning 조금 우수한 것을 위 표를 통해 확인할 수 있다. 이는 Post-Pruning 방식이 Full-tree를 생성한 후 다시 최적의

구조를 찾아 의사결정나무를 단순화시키는 방식이기에 Full-tree를 생성하지 않고 가지치기를 하는 Pre-pruning 방식보다 성능이 높을 가능성이 높기 때문이라고 해석할 수 있다. 그러나 위 표에서 나타난 두 모델의 전체적인 분류 성능의 차이가 유의미한 수준으로 높지 않다고 판단하여 모델의 복잡도가 좀 더 낮은 4-2)의 Pre-pruning 모델이 최적의 모델이라고 판단했다.



5-2) 3가지 대표적인 규칙들

3가지 대표적인 규칙들을 선정 시 entropy가 낮음과 동시에 충분한 수의 sample을 가지고 있는 것을 기준으로 아래와 같은 3가지 대표적인 규칙들을 확인할 수 있었다.

JoiningYear <= 1.308 PaymentTier <= -0.353 JoiningYear <= 0.771 PaymentTier <= -2.135 Education <= 1.5이면 class 1(떠남)로 분류된다.

JoiningYear <= 1.308 PaymentTier > -0.353 Education <= 1.5 Gender > 0.5 City_Bangalore <= 0.5 class 0(떠나지 않음)으로 분류된다.

JoiningYear <= 1.308 PaymentTier > -0.353 Education > 1.5 Education > 2.5 Age <= -0.392 class 0(떠나지 않음)으로 분류된다.

[Q6] 최적의 NN

(Neural Network) 동일한 데이터셋에 대하여 Neural Network 학습을 위해 필요한 최소 3가지 이상의 하이퍼파라미터를 선정하고, 각 하이퍼파라미터마다 최소 3개 이상의 후보 값(최소 9가지 조합)을 사용 하여 grid search를 수행한 뒤, 검증데이터에 대한 AUROC 기준으로 최적의 하이퍼파라미터 조합을 찾아 보시오.

6-1) Hyperparameter 종류 및 탐색 범위 설정

이름	설명
Hidden_layer_sizes	신경망에서 은닉층의 크기를 지정하는 하이퍼파라미터
activation	활성화함수 선택하는 하이퍼파라미터
solver	가중치의 최적화에 사용되는 알고리즘을 결정하는 하이퍼파라미터
learning_rate	학습 속도를 결정하는 하이퍼파라미터
max_iter	학습 알고리즘에서 수행할 최대 반복 횟수를 지정하는 하이퍼파라미터

탐색할 5가지의 하이퍼파라미터의 범위는 아래와 같이 설정하고 탐색을 진행하였다.

```
param_grid_q6 = {'hidden_layer_sizes': [(50,), (100,), (200,)],
                 'activation': ['relu', 'tanh', 'logistic'],
                 'solver': ['adam', 'sgd', 'lbfgs'],
                 'learning_rate':['constant', 'invscaling', 'adaptive'],
                 'max_iter':[100,200,500]}
```

Hidden_layer_sizes를 통해 모델의 복잡도와 capacity를 통제한다.

Activation의 Relu는 rectified linear unit을 나타내고, tanh은 hyperbolic tangent, logistic은 sigmoid function을 나타낸다.

Solver의 Adam은 경우 많은 문제에 적용할 수 있는 adaptive learning rate 최적화 알고리즘이다. Sgd는 large dataset에 적합한 각 훈련 뒤 가중치를 incrementally하게 update하는 알고리즘이다. Lbfgs는 quasi-newton method와 관련된 optimizer이다.

learning_rate를 통해 가중치 업데이트 크기를 조절하여 학습 속도와 수렴 속도를 조절한다.

6-2) 검증 데이터 AUROC 기준 최적의 하이퍼파라미터 조합

```
param_grid_q6 = {'hidden_layer_sizes': [(50,), (100,), (200,)],
                 'activation': ['relu', 'tanh', 'logistic'],
                 'solver': ['adam', 'sgd', 'lbfgs'],
                 'learning_rate':['constant', 'invscaling', 'adaptive'],
                 'max_iter':[100,200,500]}


best_auroc=0.0
best_model_ann=None

for params in ParameterGrid(param_grid_q6):
  model=MLPClassifier(**params, random_state=12345)
  model.fit(x_train, y_train)
  y_val_=model.predict(x_val)
  auroc=roc_auc_score(y_val, model.predict_proba(x_val)[:,1])
  if auroc>best_auroc:
    best_auroc=auroc
    best_model_ann=model

y_test_pred=best_model_ann.predict(x_test)
test_auroc_ann=roc_auc_score(y_test, best_model_ann.predict_proba(x_test)[:,1])

print(best_model_ann.get_params())
print(test_auroc_ann)
```

위 코드를 이용하여 검증 데이터의 AUROC기준 최적의 하이퍼파라미터 조합은 아래와 같이 찾을 수 있었다.

```
{'activation': 'relu', 'hidden_layer_sizes': (100,), 'learning_rate':
'constant', 'max_iter': 100, 'solver': 'adam'}
```

이 때의 AUROC 값은 0.852619가 나왔다.

[Q7] DT, NN, LR

[Q3]에서 선택한 최적의 Pre-pruning Decision Tree 모델과 [Q6]에서 선택한 최적의 Neural Network, 그리고 로지스틱 회귀분석을 사용하여 학습 데이터를 학습한 뒤, 테스트 데이터에 적용한 결과를 아래의 Confusion Matrix와 같이 작성하고 이에 대한 결과를 해석해 보시오.

[Q3]에서 구한 최적의 Pre-pruning DT 모델과 (2) [Q6]에서 선택한 최적의 NN에서 했던 작업과 동일하게 로지스틱 회귀분석도 penalty, C, solver, max_iter의 하이퍼파라미터 값을 변화시키며 그리드 서치를 통해 검증데이터를 적용한 AUROC가 가장 높은 최적의 로지스틱 회귀분석 모델을 찾았다.

```
param_grid_q7 = {'penalty': ['l1', 'l2'],
                 'C': [0.001, 0.01, 0.1, 1, 10],
                 'solver':['liblinear', 'saga'],
                 'max_iter':[100, 200, 500]
                 }


best_auroc_logi=0.0
best_model_logi=None

for params in ParameterGrid(param_grid_q7):
  model=LogisticRegression(**params, random_state=12345)
  model.fit(x_train, y_train)
  y_val_logi=model.predict(x_val)
  auroc=roc_auc_score(y_val, model.predict_proba(x_val)[:,1])
  if auroc>best_auroc_logi:
    best_auroc_logi=auroc
    best_model_logi=model

y_test_pred=best_model_logi.predict(x_test)
test_auroc_logi=roc_auc_score(y_test, best_model_logi.predict_proba(x_test)[:,1])

print(best_model_logi.get_params())
print(test_auroc_logi)
```

위 코드를 이용해 찾은 최적의 하이퍼파라미터 조합은 아래와 같았다.
`'C': 10,  'max_iter': 100, 'penalty': 'l1', 'solver': 'saga'`

이 때의 AUROC 값은 0.734771이 나왔다.

이를 바탕으로 테스트 데이터에 대해 로지스틱 회귀분석을 실행하였다. Logistic Regression, Decision Tree, Neural Network를 실행한 평가 지표는 다음과 같았다. (Decision Tree의 평가 지표는 [Q3]에서 이미 구한 지표를 표에 옮겼다.)

	TPR	TNR	ACC	BCR	F1
Logistic Regression	0.462555	0.887473	0.749284	0.640707	0.545455
Decision Tree	0.621145	0.944798	0.839542	0.766066	0.715736
Neural Network	0.590308	0.963907	0.842407	0.754322	0.708995

-LR 모델의 경우 DT 모델, NN모델과 비교하여 전체적인 성능이 떨어진다. 이는 선형 모델이기에 유연한 두 모델과 비교하여 데이터의 복잡한 패턴을 포착하기 상대적으로 어렵기 때문이라고 해석된다.

-DT 모델, NN모델 모두 우수한 성능을 보였다. [Q7]에서 사용한 DT 모델은 Pre-Pruning 모델로 트리 생성 시 사전에 가지치기 작업을 수행함으로써 모델의 복잡도를 낮추고, 과적합을 방지하는 모델이다. 이 과정에서 다양한 split 기준을 고려함으로써 feature 간 비선형 관계와 다양한 상호작용을 보여줄 수 있는 유연함을 보이고, 따라서 우수한 성능을 보였다고 해석할

수 있다. NN은 복잡한 종속성을 포착할 수 있는 여러 개의 상호 연결된 노드 계층으로 구성되기에 비선형 관계를 표현하는 능력이 뛰어나다. 따라서 데이터의 비선형 관계를 포착하는 능력이 뛰어난 DT 모델, NN 모델이 높은 성능을 보인 것이다.

-세 모델 모두 TPR 값이 TNR 값보다 유의미한 수준으로 낮았다. 이는 모델의 종류에 상관없이 클래스에 속해 있지 않는 데이터에 대한 예측을 클래스에 속해 있는 데이터에 대한 예측보다 잘한다고 해석할 수 있다. 이는 앞서 [Q1]에서 살펴봤듯이 클래스에 속해 있지 않은 데이터의 수가 클래스에 속해 있는 데이터 수보다 약 3배 정도 많아 모델 학습 시 클래스에 속해 있지 않은 데이터가 더 많이 학습되었기 때문이라고 추측된다.

[Q8] 예측 성능이 중요한 데이터셋 선정

이번에는 본인이 생각하기에 "예측 정확도"가 "예측 결과물에 대한 해석"보다 훨씬 더 중요할 것으 로 생각되는 분류 문제를 다루고 있는 데이터셋을 1개 선정하고 선정 이유를 설명하시오. 이외 가이드라 인은 [Q1]의 가이드라인과 동일합니다.

데이터링크: https://www.kaggle.com/datasets/mssmartypants/rice-type-classification

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18185 entries, 0 to 18184
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id               18185 non-null  int64
 1   Area             18185 non-null  int64
 2   MajorAxisLength  18185 non-null  float64
 3   MinorAxisLength  18185 non-null  float64
 4   Eccentricity     18185 non-null  float64
 5   ConvexArea       18185 non-null  int64
 6   EquivDiameter    18185 non-null  float64
 7   Extent           18185 non-null  float64
 8   Perimeter        18185 non-null  float64
 9   Roundness        18185 non-null  float64
 10  AspectRation     18185 non-null  float64
 11  Class            18185 non-null  int64
dtypes: float64(8), int64(4)
memory usage: 1.7 MB
None
```

본 데이터셋은 'Rice-type classification dataset'으로 쌀의 넓이, 둘레, 둥근 정도 등 쌀의 속성과 쌀의 Class를 담고 있는 dataset이다. 종속변수인 'Class'와 독립변수 9개(id는 단순 번호를 기입한

것이므로 독립변수에서 제외)를 합한 총 11개의 변수로 구성되어 있으며 데이터의 개수는 18185개로 구성되어 있다. 특별히 종속변수인 'Class'의 경우 1, 0 2개의 Class로 이루어져 있기에 binary classification 문제이다.

앞서 [Q1]에서 살펴본 데이터셋은 회사 이탈률에 관한 데이터셋으로 독립변수인 교육 수준, 입사년도 등을 통해 회사 이탈 여부를 예측하는 것보다 분석 과정에서 직원들이 회사를 이탈하는데 어떤 요인이 관여하는지 유의미한 특성을 추출하는데 그 의의가 있었다. 즉, 결과 해석을 통해 기업 경영을 개선하는 등 실질적인 목적이 보다 크다고 생각되었다.

반면에 쌀의 품종 분류 문제의 경우 독립 변수인 둘레, 둥근 정도 등에서 유의미한 특성을 추출하고 결과를 해석하는 것보다 쌀의 품종을 올바르게 예측하는 것이 더욱 중요하다고 생각한다. 이유는 이미 쌀의 품종에 관한 특성 연구는 대부분 진행되었을 것이고, 품종 자체는 이미 정해져있기 때문이다. 뿐만 아니라 실질적으로 산업 현장에서 쌀의 품종 특성에 관한 직접적인 연구보다는 정확한 품종 분류가 더 경제적이고, 많이 쓰인다고 생각이되어 쌀의 품종을 정확하게 예측하는 것은 매우 중요하다고 할 수 있다.

학습: 검증: 테스트 데이터셋의 분배 비율로 7:1.5:1.5의 비율을 사용하였다. 예측 성능이 좋아야하기에 학습데이터의 비중을 최대한으로 확보하면서 동시에 최적의 하이퍼파라미터 설정 및 테스트 데이터를 예측하는데 적절한 양의 데이터도 확보하려고 하였다. 그래서 보편적으로 많이 쓰이는 분배 비율인 6:2:2 비율에서 학습데이터의 비중을 약간 늘리고, 검증 데이터셋과 테스트 데이터셋의 비중을 동일하게 가져간 7:1.5:1.5 비율을 사용하였다.

[Q9] Pre-pruning, NN

(Decision Tree/Neural Network 공통) [Q8]에서 선택한 데이터셋을 사용하여 [Q3]에서 수행한 최적 의 pre-pruning Decision Tree 모델 찾기, [Q6]에서 수행한 최적의 Neural Network 모델 찾기를 동일하 게 수행하시오.

Decision Tree로 분류하기에 앞서 결측치 제거, 명목형 변수 원핫인코딩, scaling 작업을 수행하려고 하였다.

```
id                0
Area              0
MajorAxisLength   0
MinorAxisLength   0
Eccentricity      0
ConvexArea        0
EquivDiameter     0
Extent            0
Perimeter         0
Roundness         0
AspectRation      0
Class             0
dtype: int64
```

위 그림에서 알 수 있듯이 결측치는 존재하지 않았기에 특별히 결측치 제거 작업을 수행하진 않았다. 또한 모든 변수가 int형 혹은 float형으로 명목형 변수가 존재하지 않아 따로 데이터 가공을 진행하진 않았다. 다만 id의 경우 단순 라벨링 번호로 의미 없는 변수라 판단하여 독립 변수에서 제외하고 분석을 진행하였다. 이후 모든 독립변수에 대해 Scaling을 진행하고, 최적의 DT 모델과 NN 모델 탐색하였다.

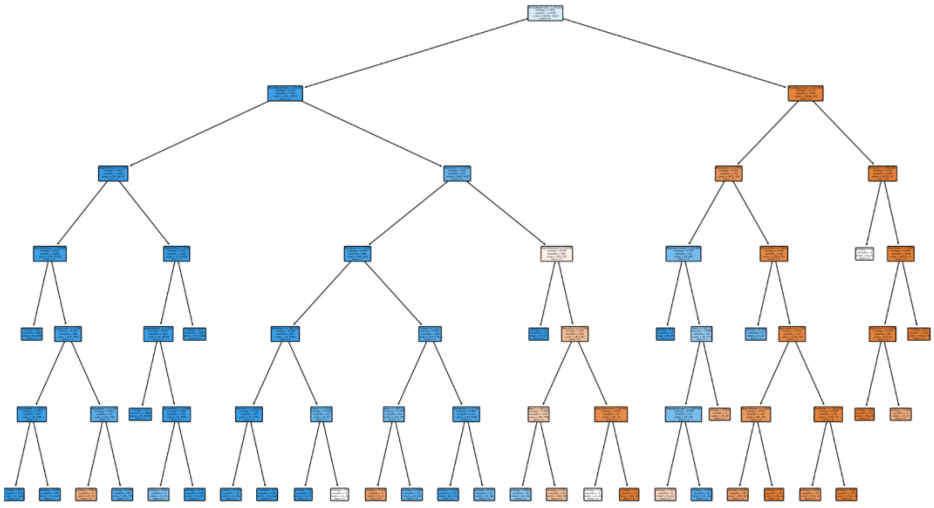
DT 모델, NN 모델의 하이퍼파라미터 종류 및 범위 설정은 [Q3], [Q6]에서 수행한 것과 동일하게 진행하였다. 단, Full tree의 깊이가 14였기에 Pre-pruning에서 max_depth 하이퍼파라미터 탐색 시 None, 3, 6, 9 네 가지의 범위를 가지고 탐색을 진행하였다. (앞서 질문들에서 하이퍼파라미터 설명을 진행하였기에 이는 생략하도록 하겠다.)

9-1)Pre-pruning

검증 데이터의 AUROC기준 최적의 하이퍼파라미터 조합은 아래와 같이 나왔다.

```
'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 4, 'min_samples_split': 2
```

이 때의 AUROC 값은 0.997292로 매우 높은 값이 나왔다.



Pre-Pruning 한 트리를 plot 한 그림은 위와 같다.
기존 Full tree 의 깊이와 리프 노드 수가 각각 14 개, 141 개에서 7 개, 29 개로 확연히 감소한 것을 확인할 수 있다. 이를 통해 트리의 복잡도가 감소했음을 확인할 수 있고, 과적합 문제를 해결함과 동시에 설명력이 증가하였다고 해석할 수 있다.

9-2)Neural Network

검증 데이터의 AUROC기준 최적의 하이퍼파라미터 조합은 아래와 같이 나왔다.

```
{'activation': 'relu', 'hidden_layer_sizes': (100,), 'learning_rate':
'constant', 'max_iter': 200, 'solver': 'lbfgs'
```

이 때의 AUROC 값은 0.998315로 매우 높은 값이 나왔다.

9-3) Full tree, DT, NN 성능 비교

	TPR	Precision	TNR	Accuracy	BCR	F1-Measure
Full Tree	0.98796	0.985324	0.982157	0.985337	0.985054	0.98664
Pre-Pruning	0.989298	0.988636	0.986212	0.987903	0.987754	0.988967
NN	0.996656	0.984798	0.981346	0.989736	0.988971	0.990691

모든 모델이 전 지표에서 굉장히 좋은 성능을 보임을 확인할 수 있다. 앞서 [Q5]에서는 Full tree 모델의 성능과 비교하여 DT, NN의 성능이 눈에 띄게 좋았다. 하지만, 이 데이터셋에서는 세 모델의 성능 차이가 무방하다고 볼 수 있다. 이유로 우선 [Q5]에서 사용한 데이터와 다르게 예측 성능이 더 중시되는 데이터이고, 데이터의 개수가 약 4배가량 더 많아 더 많은 데이터가 학습에 사용되어 성능이 더 좋다고 해석할 수 있다. 무엇보다 본 데이터셋이 실제 데이터가 아닌 분류 모델을 통한 분석을 위해 인위적으로 생성한 데이터 셋이라는 점이 가장 큰 이유라고 해석된다.

[Q10] LR, DT, NN 비교

10-1)LR

로지스틱 회귀분석도 penalty, C, solver, max_iter의 하이퍼파라미터 값을 변화시키며 그리드 서치를 통해 검증데이터를 적용한 AUROC가 가장 높은 최적의 로지스틱 회귀분석 모델을 찾았다.

이를 통해 찾은 최적의 하이퍼파라미터 조합은 아래와 같았다.

```
'C': 10, 'max_iter': 500, 'penalty': 'l1', 'solver': 'liblinear'
```

이 때의 AUROC 값은 0.998737이 나왔다.

10-2) Pre-Pruning, Neural Network, Logistic Regression 모델 분류 성능 비교

10-1)에서 구한 최적의 로지스틱 회귀분석 모델을 이용하여 테스트 데이터에 대해 로지스틱 회귀분석을 실행하였다. Logistic Regression, Decision Tree, Neural Network를 실행한 평가 지표는 다음과 같았다.

	TPR	Precision	TNR	Accuracy	BCR	F1-Measure
Full Tree	0.98796	0.985324	0.982157	0.985337	0.985054	0.98664
Pre-Pruning	0.989298	0.988636	0.986212	0.987903	0.987754	0.988967
NN	0.996656	0.984798	0.981346	0.989736	0.988971	0.990691
LR	0.996656	0.986755	0.983779	0.990836	0.990197	0.991681

DT, NN, LR 3가지의 모델 전부, 6가지의 성능 지표 모두 서로 간의 비교가 무의미할 정도로 상당히 높은 수치를 기록하였다. 이는 본 데이터셋이 실제 데이터가 아닌 분류 모델을 통한 분석을 위해 인위적으로 생성한 데이터 셋이기에 위의 confusion matrix와 같이 높은 성능을 보인 것으로 해석된다.

10-3) [Q7]과 [Q10] 각각 내에서 모델 간 비교

[Q7]에서는 3가지의 모델 중 DT모델과 NN 모델에서 가장 우수한 성능을 확인할 수 있었다. 특별히 두 모델의 경우 우열을 가리기 힘들 정도로 비슷하게 우수했다.

[Q10]에서는 DT, NN, LR 3가지의 모델 전부에서 또 모든 지표들에서 0.980을 초과할 정도로 상당히 높은 성능을 보이며, 우열을 가리기 힘들 정도로 비슷하게 우수했다.

[Q7], [Q10]을 종합적으로 고려하여 한 가지 모델만을 선정한다면 [Q7]에서도 우수한 성능을 보여준 Pre-Pruning 모델 혹은 NN 모델을 선택하겠다.

10-4) [Q7]과 [Q10]에서 데이터 셋간 전체적 성능 비교 (일치 여부 확인)

[Q7]에서는 예측 성능보다 예측 결과물에 대한 해석이 중요한 데이터셋을 다뤘고, [Q10]에서는 예측 결과물에 대한 해석보다 예측 성능이 중요한 데이터셋을 다뤘다. 따라서 [Q10]의 분류 성능이 [Q7]보다 우수할 것이라고 예상하였다.

결론부터 말하자면 예상대로 [Q10]의 분류 성능이 [Q7]보다 압도적으로 우수하였다. 다만, 데

이터셋이 달라짐에 따라 데이터의 개수, 변수의 개수, 종속 변수의 균형도 등이 다르고 이러한 요인들도 영향을 미칠 수 있기에 더 엄밀한 분석이 필요할 것으로 생각된다. 또한 [Q10]에서 사용한 데이터 셋이 인위적으로 생성한 데이터셋 대신 자연적으로 생성된 다른 데이터셋을 이용하여 분석한다면 좀 더 유의미한 분석이 될 것이라고 기대한다.