HW - Decision Tree

짱짱조

[1] ID3, CART, CHAID, C4.5 Algorithms

Decision Tree 는 분리 기준, 회귀모형/분류모형 인지, overfitting 제거/감소 technique, 불완전한 data 처리에 따라 그 종류가 나뉜다.

먼저 **ID3(Iterative Dichotomizer)**는 Ross Quinlan 에 의해 개발된 알고리즘으로 Information Gain 을 계산하여 Entropy 를 사용하여 분할한다.

CART(Classification And Regression Trees)는 반복적으로 데이터에 적용된 Gini Index 에 기반한 tree 를 생성한다. 만약 한 노드에서 모든 데이터가 한 class 로 분류된다면 purity 가 높다고 정의하며 impurity 를 줄이는 방향으로 tree 를 형성한다.

CHAID(Chi-Square Automatic Interaction Detector)은 CART 와 달리 데이터를 분할할 때 Gini 대신 Chi-Square Test 를 활용한다. 이에 따라 이 방법은 범주형 자료에 대해서만 적용 가능하다. 따라서 수치형 속성의 경우 범주형으로 변화하여 사용해야 한다는 특징이 있다.

C4.5 는 Quinlan 이 개발한 또다른 알고리즘으로 ID3 와 달리 연속형/이산형 변수 둘 다 이 알고리즘을 사용할 수 있다는 특징이 있다. 또, 불완전한 데이터를 다룰 수 있고 Decision Tree 를 pruning 하는 과정을 거쳐 Overfitting 문제를 해결하였다. 또, 훈련데이터 별로 다르게 가중치를 두어 학습할 수 있다는 특징도 있다.

우리 조가 진행한 실습에서는 Python 의 Scikit-learn Library 에서 기본으로 제공하는 CART 알고리즘을 사용하였다. 따라서 아래 기술할 각 Decision Tree 들은 모두 CART 알고리즘을 기반으로 진행되었다

[2] Regression Decision Tree

(1) Regression Decision Tree 의 개념

Decision Tree 란 데이터를 분석하여 이들 사이에 존재하는 패턴을 예측해 node 들을 규칙들의 조합으로 나타내는 구조이다. 각 데이터들은 최종 노드로 분류될 때까지 각 분기마다 특정 설명변수의 특정 값을 기준으로 분류되고 모델은 어느 노드에서 어느 설명변수를 어느 값으로 분류할 지를 찾는 과정을 거친다. Decision Tree 의 한 종류인 **Regression Decision Tree** 의 경우는 최종 node 들에 속한 데이터들에 대해 종속변수의 평균값을 결과치로 제공하게 된다. 즉, 어떠한 X 가 Decision Tree 에 들어왔을 때 해당 X 는 Decision Tree 의 규칙에 따라 특정 최종 node 에 속하게 되고, 해당 node 의 y 값 평균을 y 값으로 예측되게 된다.

(2) Dataset, Data Preprocessing

연속형 타겟 변수에 대한 Regression Decision Tree 실습에서는 한국 프로야구 타자 데이터를 이용하였다. 해당 데이터는 kaggle 에서 가져온 데이터이다.

(출처): https://www.kaggle.com/bluemumin/kbo-baseball-for-kaggle)

 # 변수 설명

다음 표는 데이터 내 변수들의 명칭과 의미를 나타낸 것이다.

변수명	batter_name	age	G	PA	AB	R	H	2B	3B
설명	선수 이름	나이	경기 수	타수	타석 수	득점	안타	2 루타	3 루타
변수명	HR	TB	RBI	SB	CS	BB	HBP	GB	SO
설명	홈런	총 루타 수	타점	도루 성공	도루 실패	볼넷 수	몸에 맞은 공	고의 4 구	삼진
변수명	GDP	BU	fly	year	salary	war	cp	tp	1B
설명	병살타	희생타	희생플라이	시즌	연봉	대체 선수 승리 기여도	최근 포지션	통합 포지션	1 루타

변수명	fFBP	avg	OBP	SLG	OPS	p_year	YAB	YOPS	year_born
설명	BB+HBP	타율	출루율	장타율	OBP+SLG	다음시즌	다음시즌 타율	다음시즌 OPS	태어난 연도

 # Data Preprocessing

우리가 사용하고자 하는 데이터에는 총 36 개의 column, 1913 개의 row 가 있다. 데이터를 pandas.Dataframe 형식으로 바꿔 첫 5 row 만 출력하면 다음과 같다.

	batter_name	age	G	PA	AB	R	H	2B	3B	HR	TB	RBI	SB	CS	BB	HBP	GB	SO	GDP	BU	fly	year	salary	war	year_born
0	백용환	24	26	58	52	4	9	4	0	0	13	3	0	0	6	0	0	16	3	0	0	2013	2500	-0.055	1989-03-20
1	백용환	25	47	86	79	8	14	2	0	4	28	10	0	0	5	0	0	28	1	2	0	2014	2900	-0.441	1989-03-20
2	백용환	26	65	177	154	22	36	6	0	10	72	30	3	1	19	1	0	47	5	0	3	2015	6000	0.783	1989-03-20
3	백용환	27	80	199	174	12	34	7	0	4	53	15	2	1	19	1	1	52	6	3	2	2016	6000	-0.405	1989-03-20
4	백용환	28	15	20	17	2	3	0	0	0	3	1	0	0	3	0	0	3	2	0	0	2017	5500	-0.130	1989-03-20

같은 선수에 대하여, 연도별로 row 가 존재한다는 것을 알 수 있다. 이 데이터에서 타자 이름이나 생년과 같이 분석에 불필요하거나 결측치를 포함하는 column 4 개를 제외하여 최종적으로 32 개의 column 을 사용하였다. 최종 32 개의 column 은 전부 continuous variable 이라 one-hot encoding 과 같은 추가적인 전처리는 해주지 않았다.

전처리를 마친 데이터에서 타자의 연봉을 나타내는 'salary' column 을 y 로 지정하면 다음과 같다.

```
0        2500
1        2900
2        6000
3        6000
4        5500
        ...
1908    30000
1909     3100
1910     6200
1911    50000
1912    50000
Name: salary, Length: 1885, dtype: int64
```

다음은 해당 데이터셋의 기초통계량 관련 지표이다. (변수가 너무 많은 관계로 기타 부분은 코드 참조)

	age	G	PA	AB	R	H	2B	3B	HR	TB
count	1885.000000	1885.000000	1885.000000	1885.000000	1885.000000	1885.000000	1885.000000	1885.000000	1885.000000	1885.000000
mean	26.929973	80.361273	265.480637	231.707692	34.473740	64.928382	11.491247	1.114589	6.451459	98.003183
std	4.530901	43.730725	194.986358	169.735610	29.100893	52.812598	10.168305	1.768930	8.481615	84.548834
min	18.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	24.000000	42.000000	68.000000	62.000000	8.000000	14.000000	2.000000	0.000000	0.000000	19.000000
50%	27.000000	92.000000	259.000000	226.000000	29.000000	59.000000	10.000000	0.000000	3.000000	82.000000
75%	30.000000	119.000000	443.000000	386.000000	55.000000	109.000000	19.000000	2.000000	9.000000	163.000000
max	41.000000	144.000000	672.000000	576.000000	135.000000	201.000000	47.000000	17.000000	54.000000	373.000000

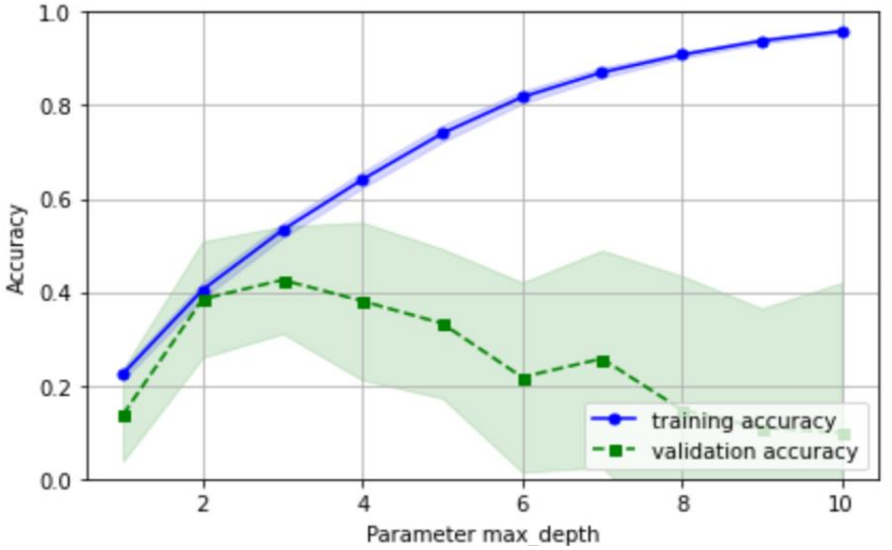
 # Training & Test Data 분리

해당 데이터셋을 사용할 때 전부 train data 로 사용하는 것이 아니라 sklearn 에서 제공하는 툴을 이용해 데이터를 8:2 의 비율로 분리해서 사용하였다.

```
from sklearn.model_selection import train_test_split
train_x, test_x, train_y, test_y = train_test_split(x, y, test_size = 0.2, random_state = 42)
```

(3) 결과 분석, 성능 비교

Tree 에는 여러가지 제약 조건을 설정할 수 있는데 그 중 max_depth 는 tree 가 root node 에서 terminal node 까지 거치는 최대 노드의 수, tree 의 최대 depth 를 의미한다. 우선 max_depth 를 조절하며 여러 가지의 tree 를 만들어보았다.



위 그래프와 같은 결과를 얻을 수 있었다. 위 그래프를 해석하면 depth 가 깊은 decision tree 일수록 세부적인 규칙을 더 적용할 것이고, 이러한 세부적인 규칙은 대개 train data 에만 fit 한 규칙을 생성하곤 한다. 그렇기에 결과적으로 train accuracy 의 경우에는 depth 가 깊어질수록 좋은 성능을 내지만 validation accuracy 에서는 성능이 증가하다 다시 떨어지는 과적합(overfitting) 현상을 볼 수 있다.

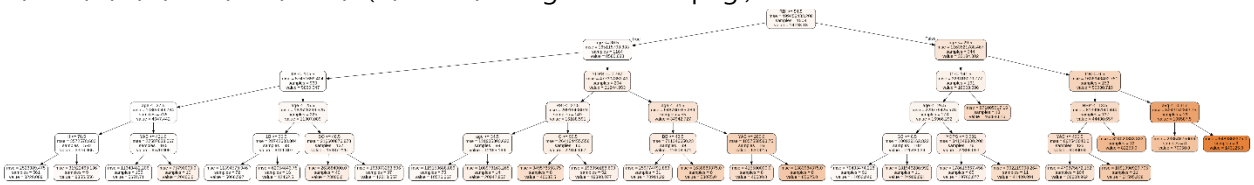
하지만 Decision Tree 를 생성하는 데 사용하는 hyperparameter 에는 max_depth 만 있는 것이 아니다. 리프노드가 되기 위해 필요한 최소한의 샘플 데이터수를 설정하는 min_samples_leaf 등의 hyperparameter 도 존재하고 근본적인 criterion 을 mse 로 할지, mae 로 할지 정하는 parameter 도 존재한다. 그렇기에 우리는 sklearn 에서 제공하는 GridSearchCV 를 활용해 위에서 설명한 3 가지 hyperparameter 를 조절하며 hyperparameter tuning 을 시도해보았다.

```
param_range1 = [1,2,3,4,5,6,7,8,9,10]
param_range2 = [1,2,4,8,16,32]
param_range3 = ['mse', 'mae']
```

각 params 의 범위는 위와 같고 해당 parameter 들에 대해서 연산을 진행해보았을 때, 아래와 같은 결과와 최적 parameter 들을 찾을 수 있었고 이를 기반으로 decision tree 를 생성하였다.

```
278606767.08326125
{'decisiontreeregressor__criterion': 'mse', 'decisiontreeregressor__max_depth': 5, 'decisiontreeregressor__min_samples_leaf': 8}
```

이를 시각화하면 다음과 같다. (파일 첨부: 'Regression DT.png')

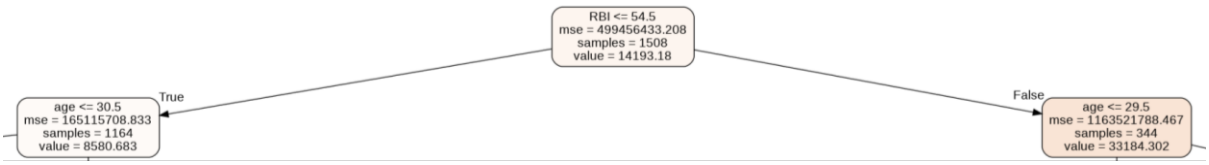


어떠한 hyperparameter 도 tuning 하지 않은 채로, sckit-learn 에서 default 로 제공되는 parameter 들과 사용했을 때는 R squared: 0.328 로 상당히 낮은 값을 기록했지만 hyperparameter tuning 을 거쳐 찾아낸 최적의 tree 는 R squared: 0.611 로 성능이 굉장히 향상됨을 확인할 수 있었다.

최적화 전	최적화 후
R squared: 0.328 MSE: 368385941.645	R squared: 0.611 MSE: 213107153.377

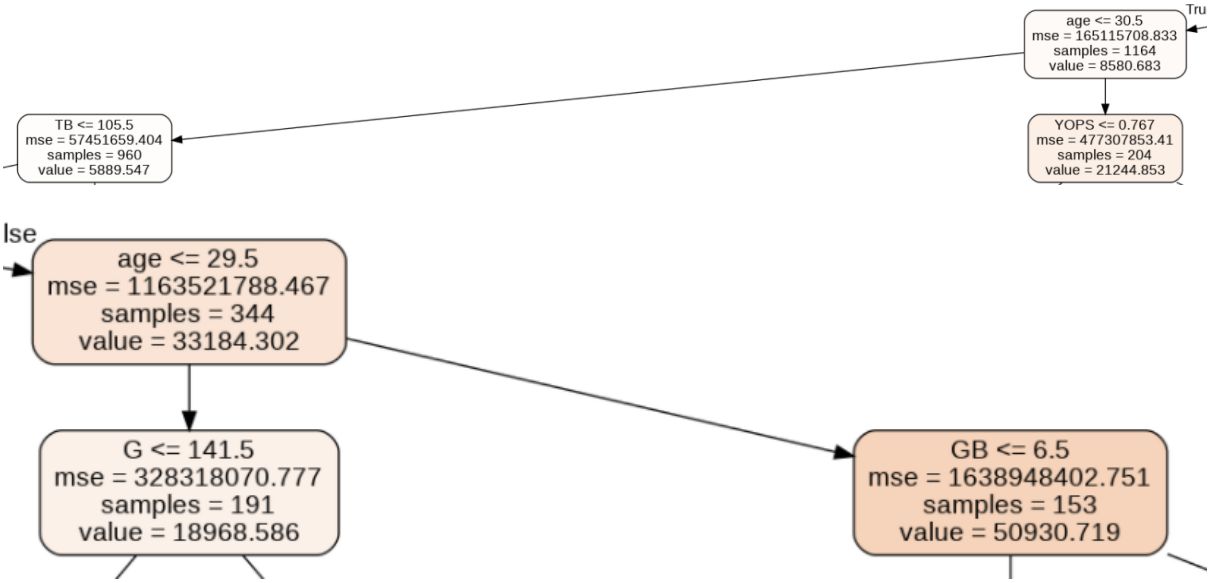
(4) 규칙 해석

 # Depth 1



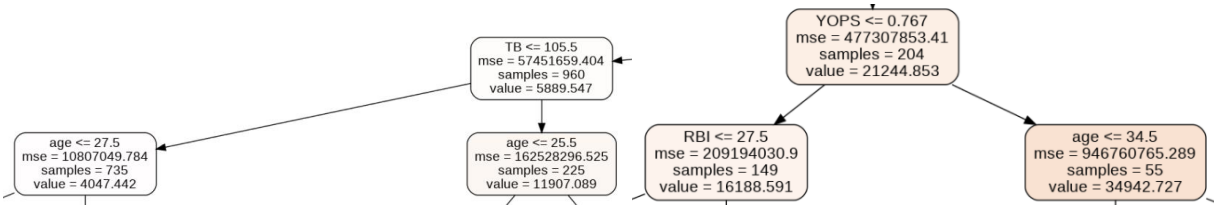
Root node 는 RBI(타점) 값이 54.5 이하인지를 기준으로 분기한다. 타점이 높은 그룹은 낮은 그룹보다 salary(연봉)가 약 4 배가량 높았으며 그 수는 약 1/3 수준이였다. 이 단계에서 타점이 가장 중요한 요소임을 유추할 수 있다.

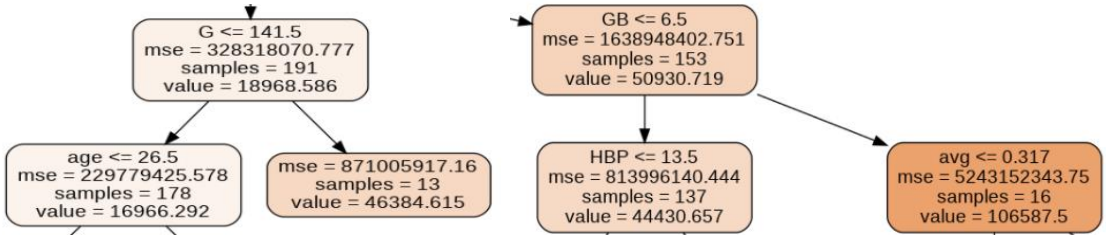
 # Depth 2



다음 depth 에서는 양쪽 모두 age(나이)를 기준으로 분기한다. 약 30 세 이하인 그룹은 그렇지 않은 그룹보다 연봉이 3~4 배가량 높았다. 이 단계에서 나이가 두 번째로 주요한 요소이며 젊은 선수의 연봉이 높은 경향을 확인할 수 있다.

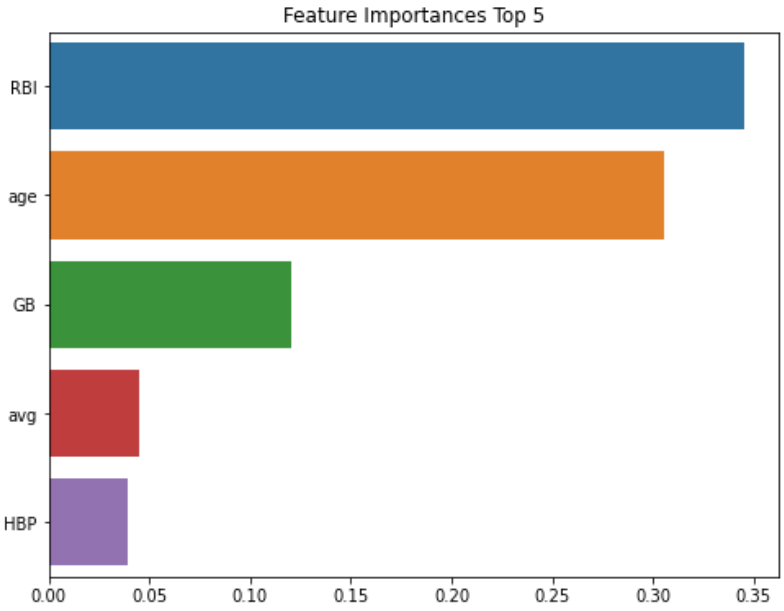
 # Depth 3





세 번째 depth 에서는 TB(총 루타 수), YOPS(다음 시즌 OPS; 출루율 + 장타율), G(게임 수), GB(고의 4 구)의 다양한 기준으로 분기하였다. 총 루타 수가 105.5 초과, YOPS 가 0.767 초과, 게임 수가 141.5 초과, 고의 4 구가 6.5 초과인 그룹이 반대 그룹보다 약 2~3 배의 연봉을 받는다는 사실을 확인할 수 있다.

(5) Feature Importance



앞서 규칙 해석 단계에서 확인했듯이 RBI(타점)과 age(나이)가 가장 주요한 feature 로 작용했음을 확인할 수 있다. 특히 둘의 중요도 합이 65% 이상을 차지하여, 야구선수 연봉을 결정하는 데 있어 가장 중요한 지표임을 확인할 수 있다. 다음으로는 세 번째 depth 에서 기준으로 사용되었던 GB(고의 4 구)가 뒤를 이었다.

[3] Classification Decision Tree

(1) Classification Decision Tree 의 개념

Decision Tree 의 다른 한 종류인 Classification Decision Tree 는 Regression Decision Tree 와 다르게 최종 node 들에 속한 데이터들에 대해 종속변수의 category 를 결과치로 제공한다. 즉, 어떠한 X 가 Decision Tree 에 들어왔을 때 해당 X 는 Decision Tree 의 규칙에 따라 특정 최종 node 에 속하게 되고, 해당 node 의 category 를 예측한다.

(2) Dataset, Data Preprocessing

본 실습에서는 Kaggle 에 있는 'Universal bank data for classification' 데이터를 이용하여 고객이 개인 융자(Personal Loan)를 받았는지 여부를 분류하고자 한다.
(출처: https://www.kaggle.com/sriharipramod/bank-loan-classification)

 # 변수 설명

다음 표는 데이터 내 변수들의 명칭과 의미를 나타낸 것이다.

ID	Customer ID
Age	Customer's age in completed years
Experience	# years of professional experience
Income	Annual income of the customer ($1,000)
ZIPCode	Home Address ZIP code
Family	Family size of the customer
CCAvg	Average spending on credit cards per month ($1,000)
Education	Education Level 1: Undergrad, 2: Graduate, 3: Advanced/Professional
Mortgage	Value of house mortgage if any ($1,000)
Securities Account	Does the customer have a securities account with the bank?
CD Account	Does the customer have a certificate of deposit (CD) account with the bank?
Online	Does the customer use internet banking facilities?
CreditCard	Does the customer use a credit card issued by Universal bank?
Personal Loan	**Did the customer accept the personal loan offered in the last campaign?**

위 변수들의 형태별 분류는 아래와 같다.

1) 연속형 변수: ID, Age, Experience, Income, ZIPCode, Family, CCAvg

2) 이진 변수 : Securities Account, CD Account, Online, Credit Card, Personal Loan

(Y – 1: accet personal loan, 0: do not accept personal loan)

3) 범주형 변수 : Education – 3 가지 범주

 # 사용할 변수 정리

```
## Drop columns which are not significant
bank_data.drop(["ID","ZIP Code"],axis=1,inplace=True) # 필요없는 column ID, ZIP Code는 버린다.
bank_data = bank_data [['Age', 'Experience', 'Income', 'Family', 'CCAvg', 'Education', 'Mortgage',
                        'Securities Account', 'CD Account', 'Online', 'CreditCard', 'Personal Loan']]
```

총 12 가지의 변수들 중 ID 와 ZIP Code 의 경우 개인 융자 여부 판단에 영향을 미치지 않으므로 두 column 을 제외해주고 column 의 순서를 정리해 'Personal Loan' column 이 마지막에 위치하도록 조정해준다.

 # One Hot Encoding (['Education'])

'Education' 변수의 경우 1,2,3 단계로 categorical 하게 나눠지는 변수이다. 0 은 undergraduate 를, 1 은 graduate 를, 2 는 Advanced/Professional 을 의미한다. Categorical 한 변수의 경우 모두 One hot encoding 처리를 해주어야 한다. 따라서, get_dummies()를 활용해 Education 변수를 Education_1, Education_2, Education_3 변수로 변환시키는 작업을 해준다. 아래를 변환시켜준 결과이다.

	Age	Experience	Income	Family	CCAvg	Mortgage	Securities Account	CD Account	Online	CreditCard	Education_1	Education_2	Education_3
0	25	1	49	4	1.6	0	1	0	0	0	1	0	0
1	45	19	34	3	1.5	0	1	0	0	0	1	0	0
2	39	15	11	1	1.0	0	0	0	0	0	1	0	0
3	35	9	100	1	2.7	0	0	0	0	0	0	1	0
4	35	8	45	4	1.0	0	0	0	0	1	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
4995	29	3	40	1	1.9	0	0	0	1	0	0	0	1
4996	30	4	15	4	0.4	85	0	0	1	0	1	0	0
4997	63	39	24	2	0.3	0	0	0	0	0	0	0	1
4998	65	40	49	3	0.5	0	0	0	1	0	0	1	0
4999	28	4	83	3	0.8	0	0	0	1	1	1	0	0

Education_1 값이 1 일 경우 학업 수준이 Undergraduate 에 속하고, Education_2 가 1 일 경우 Graduate 에, Education_3 가 1 일 경우 고객이 Advanced/Professional 에 속한다는 것을 의미한다.

 # Training & Test Data 분리

```
from sklearn.model_selection import train_test_split
train_features, test_features , train_target, test_target = train_test_split(
    b_features, b_target, test_size = 0.2, random_state = 2020, stratify=b_target)
```

총 5000 개의 데이터 중 train data 와 test data 를 8:2 의 비율로 나눠, train data 4000 개와 test data 1000 개로 분리한다.

 # 불균형 데이터 처리

```
import pandas as pd
pd.DataFrame(train_target)['Personal Loan'].value_counts()
```

```
0    3616
1     384
Name: Personal Loan, dtype: int64
```

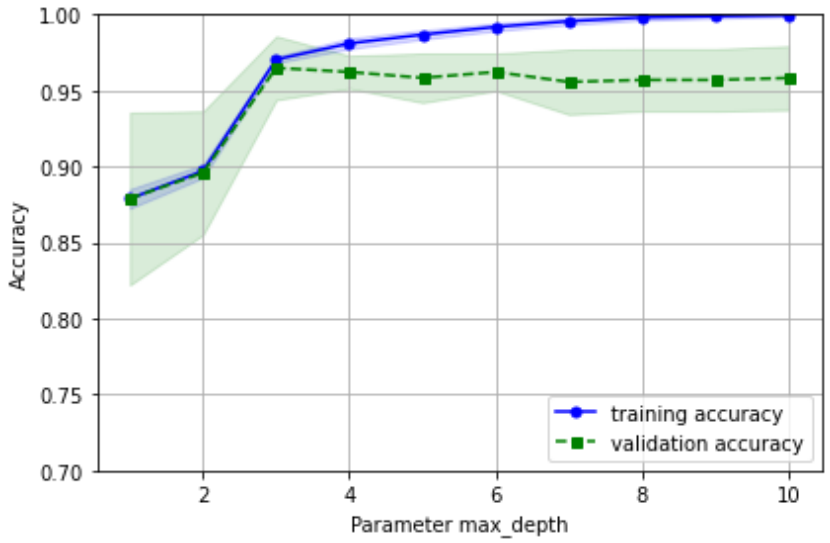
```
# Under Sampling : Y값을 {0,1} 각각 384명씩
## sampling하기 전에 shuffling을 해주기(행 순서 섞기)
import sklearn
x_shuffled = sklearn.utils.shuffle(train_features, random_state=2020)
y_shuffled =sklearn.utils.shuffle(train_target, random_state=2020)

import imblearn
from imblearn.under_sampling import RandomUnderSampler
train_features_us, train_target_us = RandomUnderSampler(random_state=2020).fit_resample(x_shuffled, y_shuffled)
```

4000 개의 train data 중 개인융자를 받은 고객 데이터는 384 개인 것에 비해 개인융자를 받은 고객의 데이터는 3616 개로 편향되어 있는 것을 확인할 수 있다. 따라서 Undersampling 을 진행해 개인 융자를 받은 고객과 받지 않은 고객 데이터 수를 384 개로 균형을 맞추었다.

(3) 결과 분석, 성능 비교

Tree 에는 여러가지 제약 조건을 설정할 수 있는데 그 중 max_depth 는 tree 가 root node 에서 terminal node 까지 거치는 최대 노드의 수, tree 의 최대 depth 를 의미한다. 우선 max_depth 를 조절하며 여러 가지의 tree 를 만들어보았다.



Depth 가 깊은 Decision Tree 일수록 세부적인 규칙을 더 적용할 것이다. 하지만 규칙이 세부적일수록 Train Data 에만 fit 한 Tree 가 나올 것이고, 결과적으로 training accuracy 는 올라가지만 validation accuracy 는 떨어지는 결과가 나올 것이다. (Overfitting)

이를 방지하고 최적의 tree 를 생성하기 위해 hyperparameter tuning 을 시도하여 최적화를 진행하였다. Sklearn 의 GridSearchCV 를 활용해 max_depth, min_samples_leaf, criterion 를 조절하였고 각 parameter 의 범위는 다음과 같다.

```
param_range1 = [1,2,3,4,5,6,7,8,9,10]
param_range2 = [10, 20, 30, 40, 50]
param_range3 = ['gini', 'entropy']
```

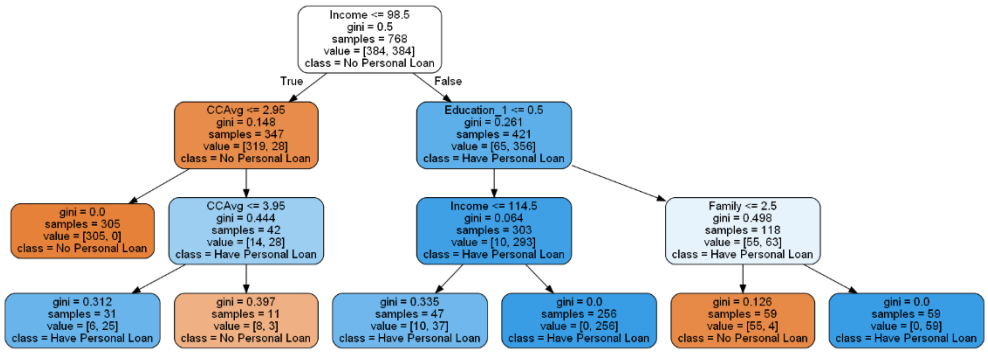
해당 parameter 들에 대해서 연산을 진행해보았을 때, 다음과 같은 결과와 최적의 parameter 를 찾을 수 있었고, 이를 기반으로 decision tree 를 생성하였다.

```
# 최적의 모델 선택

best_tree = gs.best_estimator_ # 최적의 파라미터로 모델 생성
best_tree.fit(train_features_us, train_target_us)
```

```
Pipeline(steps=[('decisiontreeclassifier',
                 DecisionTreeClassifier(max_depth=3, min_samples_leaf=10,
                                        random_state=2020))])
```

이를 시각화하면 다음과 같다. (첨부: 'Categorical DT.png')



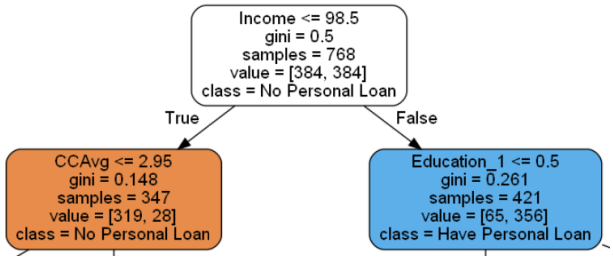
최적화 전후 성능 지표들을 확인하면 다음과 같다.

최적화 전	최적화 후
정확도 accuracy: 0.960	정확도 accuracy: 0.944
정밀도 precision: 0.719	정밀도 precision: 0.637
재현율 recall: 0.958	재현율 recall: 0.969
F1-score: 0.821	F1-score: 0.769
AUC: 0.959	AUC: 0.955

최적화 전후의 성능 지표가 크게 차이가 나는 것은 아니지만 최적화를 진행했기에 최적화 후의 tree 가 일반화가 더 잘 된 것이라고 볼 수 있다.

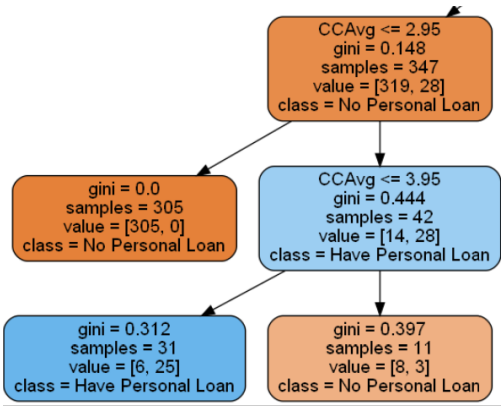
(4) 규칙 해석

 # Depth 1



Root node 는 Income(소득)을 기준으로 분기한다. 연소득이 $98,500 보다 낮으면 Personal Loan 을 받지 않았고, 높으면 Personal Loan 을 받은 것으로 볼 수 있다. Personal Loan 을 받았는지에 대한 여부에서 가장 중요한 요소는 연소득이라 생각할 수 있다.

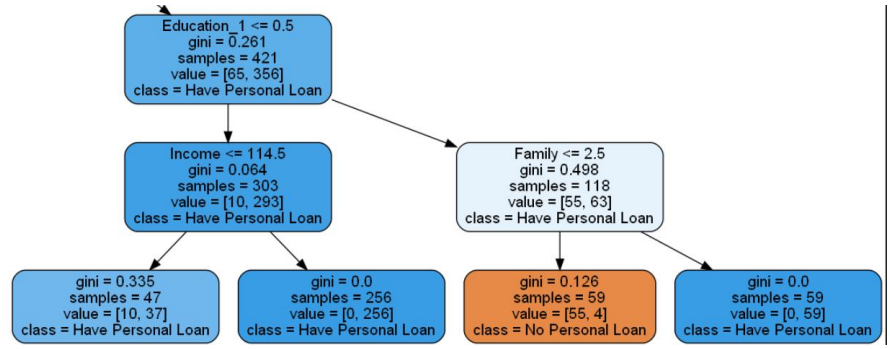
 # Left Part (Depth 2~)



두 번째 depth 에서 왼쪽 부분의 경우 CCAvg(월 평균 신용카드 사용액)을 기준으로 분기한다. 사용액이 $2,950 보다 낮으면 Personal Loan 을 받지 않았고, 높으면 받은 것으로 볼 수 있다.

그 다음 depth 에서도 CCAvg 를 기준으로 분기한다. 사용액이 $3,950 보다 낮으면 Personal Plan 을 받고, 높으면 받지 않은 것으로 볼 수 있다.

 # Right Part (Depth 2~)



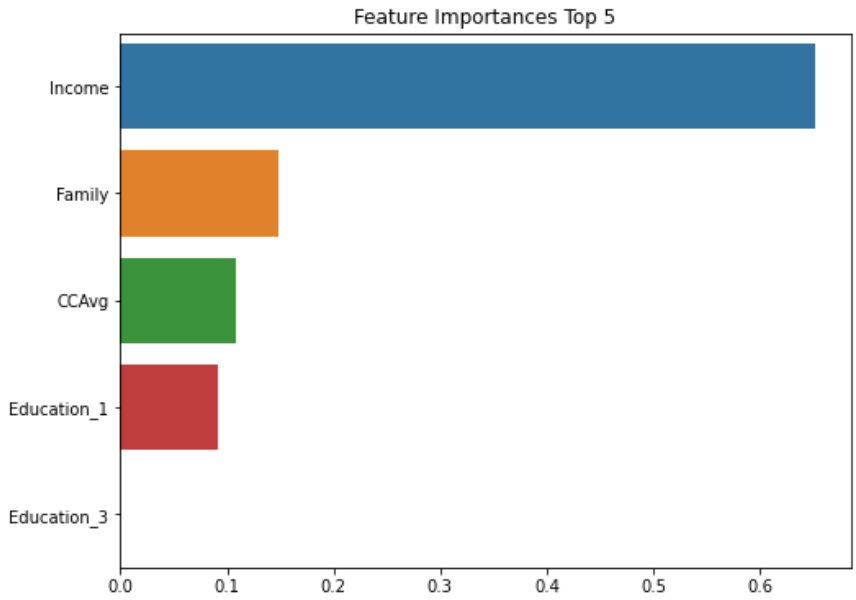
두 번째 depth 에서 오른쪽 부분의 경우 Education_1(Undergrad)를 기준으로 분기한다. Undergrad 인 경우 Personal Loan 을 받고, Undergrad 가 아닌 경우(Graduate, Advanced/Professional) Personal Loan 을 받지 않은 것으로 볼 수 있다.

그 다음 depth 에서는 Income(연소득), Family(가족 구성원 수)를 기준으로 분기한다.

연소득이 $114,500 보다 낮으면 Personal Plan 을 받지만, 높으면 받지 않는 것으로 볼 수 있다.

가족 구성원 수가 1~2 명인 경우 Personal Plan 을 받지 않지만, 3 명 이상인 경우 Personal Plan 을 받는 것으로 볼 수 있다.

(5) Feature Importance



이를 바탕으로 Features Importance Top5 를 선정했는데, Root Node 분기 기준이었던 Income(연소득)이 가장 중요한 feature 로 작용했다고 나타났다. 특히 Income(연소득)의 중요도가 60% 이상을 차지하여 Personal Loan 을 받았는지 여부에 있어 가장 중요한 feature 임을 확인할 수 있다. 다음으로는 분기 기준점이 되었던 Family(가족구성원 수), CCAvg(월 평균 신용카드 사용액), Education_1(학력-졸업 여부)이 뒤를 이었다.