











2. 의사결정나무 분석 수행

- Default of credit card clients dataset
- 의사결정나무 분석을 위하여 numerical attribute type의 데이터를 조건으로 설정하였습니다. Business area의 데이터, Attribute는 10~100개, instance는 greater than 1000을 원한다는 데이터 조건도 추가하였더니 옆 사진과 같이 14개의 데이터 셋이 나왔습니다.
- 그 중, 신용카드 고객과 관련된 데이터 셋이 흥미로운 주제라고 생각하여 선택하였습니다.

Browse Through: 14 Data Sets

Default Task	Name	Data Types
Classification (12) Regression (8) Clustering (4) Other (0)	 Bank Marketing	Multivariate
Attribute Type - Undo Categorical (0) Numerical (14) Mixed (0)	 Cargo 2000 Freight Tracking and Tracing	Multivariate, Seq
Data Type - Undo Multivariate (14) Univariate (0) Sequential (3) Time-Series (1) Text (0) Domain-Theory (0) Other (0)	 clickstream data for online shopping	Multivariate, Seq
Area - Undo Life Sciences (14) Physical Sciences (15) CS / Engineering (49) Social Sciences (3) Business (14) Game (1) Other (15)	 default of credit card clients	Multivariate
# Attributes - Undo Less than 10 (4) 10 to 100 (14) Greater than 100 (1)	 Facebook Live Sellers in Thailand	Multivariate
# Instances - Undo Less than 100 (1) 100 to 1000 (5) Greater than 1000 (14)	 Incident management process enriched event log	Multivariate, Seq
Format Type Matrix (12) Non-Matrix (2)	 Iranian Churn Dataset	Multivariate
	 Iranian Churn Dataset	Multivariate
	 Online News Popularity	Multivariate
	 Online Shoppers Purchasing Intention Dataset	Multivariate

- 의사결정나무 모델 제작
- GridSearchCV estimator로 최적화
- 필요 라이브러리, 패키지 import
- csv 데이터 불러오기
- data info, ID column 불필요하기에 제거(drop), duplicated() → 중복되는 열 있다면 제거, isna() → null 값 체크

기계학습 과제(2)

Default of Credit Card Clients data set

We will create Decision tree models in order to make predictions and compare them with the original paper.

⇒ After the initial predictions, model will be "optimized" by GridSearchCV estimator (search for the best set of hyperparameters for every model)

• Goal: Using the models we created, we will try to predict the class value of dummy column with better scores (accuracy and F1) than the scores presented in the original paper.

```
In [12]: import pandas as pd
from pandas.api.types import CategoricalDtype
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings('ignore')
import smtplib # (이메일 발송을 위한 모듈)
from smtplib import SMTP
from IPython.display import display

# 1. 데이터 로드
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
from sklearn.preprocessing import StandardScaler

# 2. 데이터 로드
from sklearn import metrics
from sklearn.metrics import f1_score, confusion_matrix, mean_squared_error, mean_absolute_error, classification_report, roc_auc_score
import matplotlib.pyplot as plt
```

Part 1: Load and clean the data

• Load the data from the csv file and check for any "important", such as null values or duplicate rows.
• If any of these will appear, we will remove them from the data set.
• We will also plot the correlations of the class column with all the other columns.

```
In [13]: # Load the data
data = pd.read_csv('default_of_credit_card_clients.csv')

Out[13]:
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_1	PAY_2	PAY_3	PAY_4	PAY_5	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	BILL_AMT5	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY_AMT4	PAY_AMT5	default
0	1	20000	2	2	1	24	2	2	2	2	2	2	0	0	0	0	0	0	0	0	0	0	0
1	2	120000	2	2	2	26	-1	2	0	0	0	0	3272	3435	3201	0	100	0	0	0	0	0	0
2	3	80000	2	2	2	34	0	0	0	0	0	0	14331	14648	10860	1016	150	0	0	0	0	0	0
3	4	50000	0	2	1	37	0	0	0	0	0	0	28314	28859	25847	2030	201	0	0	0	0	0	0
4	5	30000	1	2	1	37	-1	0	-1	0	0	0	20840	19148	19131	2000	3668	0	0	0	0	0	0
29995	29996	220000	1	3	1	39	0	0	0	0	0	0	88024	91237	15980	8000	2000	0	0	0	0	0	0
29996	29997	120000	1	3	2	43	-1	-1	-1	-1	-1	-1	3879	6185	0	1837	352	0	0	0	0	0	0
29997	29998	30000	1	2	2	37	4	3	2	-1	-1	-1	25978	25562	10387	0	0	0	0	0	0	0	0
29998	29999	80000	1	3	1	41	1	-1	0	0	0	0	52774	11855	48944	85000	340	0	0	0	0	0	0
29999	30000	30000	1	2	1	46	0	0	0	0	0	0	36035	32429	15313	2078	180	0	0	0	0	0	0

30000 rows × 25 columns

```
In [13]: # Information
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
#   Column                Non-Null Count  Dtype
---  --
0   ID                     30000 non-null   int64
1   LIMIT_BAL              30000 non-null   int64
2   SEX                   30000 non-null   int64
3   EDUCATION              30000 non-null   int64
4   MARRIAGE               30000 non-null   int64
5   AGE                   30000 non-null   int64
6   PAY_0                  30000 non-null   int64
7   PAY_1                  30000 non-null   int64
8   PAY_2                  30000 non-null   int64
9   PAY_3                  30000 non-null   int64
10  PAY_4                  30000 non-null   int64
11  PAY_5                  30000 non-null   int64
12  BILL_AMT1              30000 non-null   int64
13  BILL_AMT2              30000 non-null   int64
14  BILL_AMT3              30000 non-null   int64
15  BILL_AMT4              30000 non-null   int64
16  BILL_AMT5              30000 non-null   int64
17  PAY_AMT1               30000 non-null   int64
18  PAY_AMT2               30000 non-null   int64
19  PAY_AMT3               30000 non-null   int64
20  PAY_AMT4               30000 non-null   int64
21  PAY_AMT5               30000 non-null   int64
22  PAY_AMT6               30000 non-null   int64
23  PAY_AMT7               30000 non-null   int64
24  default payment next month  30000 non-null   int64
dtypes: int64(25)
memory usage: 5.7 MB
```

• ID column이 연속성 목적뿐이기에 제거

```
In [4]: # Drop "ID" column
data = data.drop('ID', axis=1)
```

• 중복 열 확인, 있다면 제거

```
In [5]: # Check for duplicate rows
print(f"There are {data.duplicated().sum()} duplicated rows in the data set.")
```

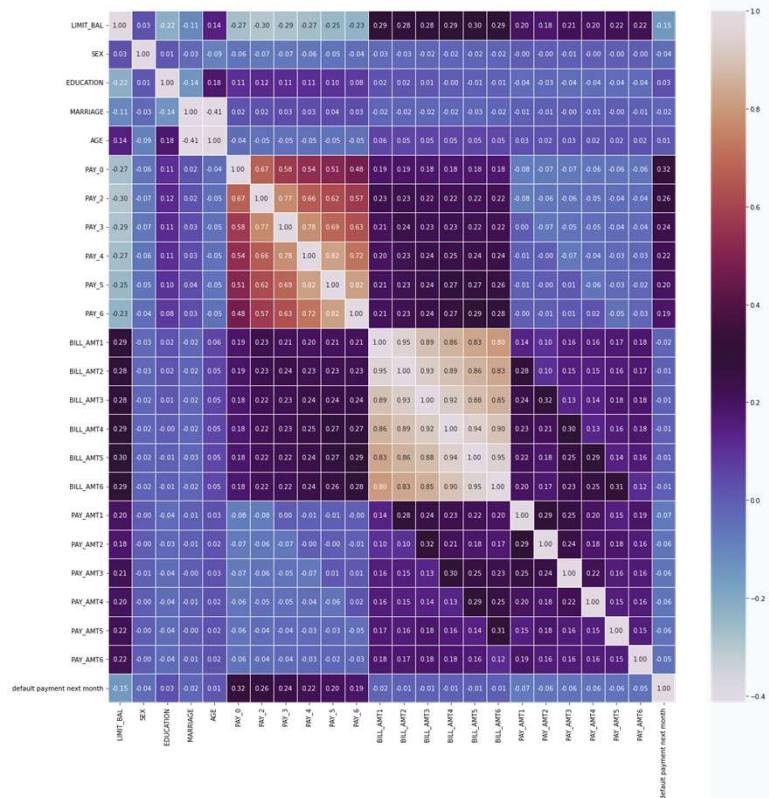
```
# Remove duplicate rows
data = data.drop_duplicates()
print("The duplicate rows are removed.")
```

There are 35 duplicated rows in the data set.
The duplicate rows are removed.

• null값 체크하기

```
In [6]: # Check for null values
print(f"There are {data.isna().any().sum()} cells with null values in the data set.")
```

There are 0 cells with null values in the data set.



- 데이터셋의 상관관계 matrix plot

In [7]: `plt.figure(figsize = (20,20))
sns.heatmap(data.corr(), annot = True, cmap = 'twilight', linewidth = 0.5, fmt = '.2f');`

- 데이터셋 상관관계 확인 위한 heatmap
- train/test set 데이터 분할, standardization, 의사결정나무 적합(fit)

Part 2: Pre-processing

- Prepare data for models.
- choose the columns that will be our independent variables and which column the class that we want to predict.
- After that, split data into train and test sets and perform a standardization upon them.

In [8]: `# Distinguish attribute columns and class column
X = data[data.columns[: -1]]
y = data['default payment next month']`

In [9]: `# Split to train and test sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 25)`

In [10]: `# Standardization
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)`

Part 3: Modeling

In this section, build models:

- Decision tree

=> Model will be trained and make a prediction for the test set.

- Accuracy, f1 score, confusion matrix and ROC will be calculated for each model.
- Then we will use the GridSearchCV module to tune our models and search for the best hyperparameters (increase the accuracy of each model)

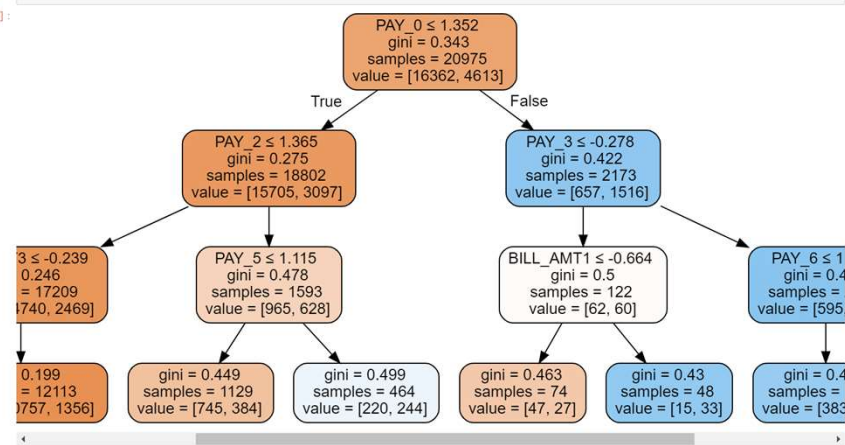
Decision tree

```
In [11]: # Initialize a decision tree estimator
tr = tree.DecisionTreeClassifier(max_depth = 3, criterion = 'gini', random_state = 25)

# Train the estimator
tr.fit(X_train, y_train)
```

```
Out[11]: DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3, random_state=25)
```

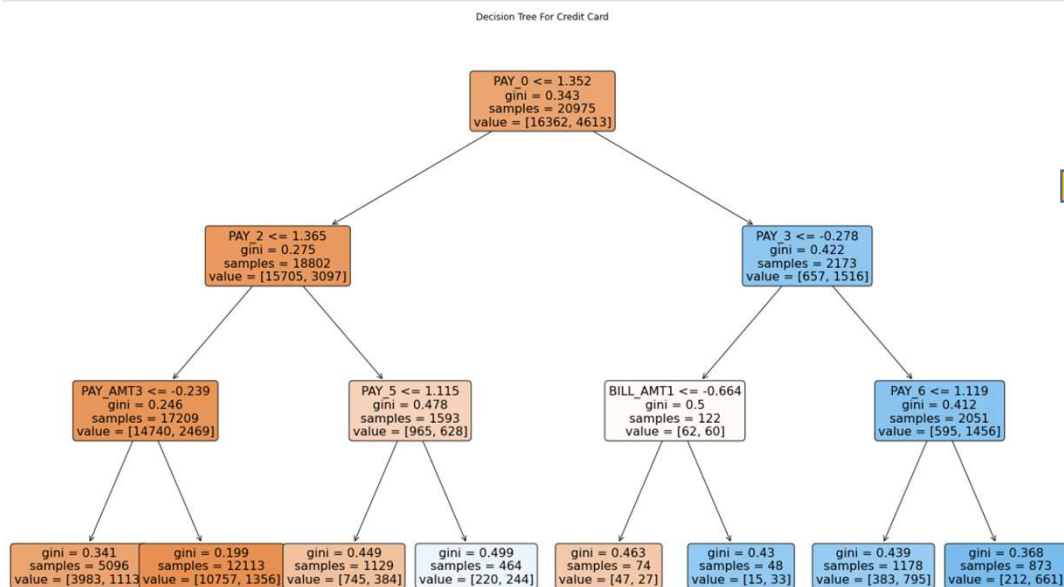
```
In [12]: # Plot the tree
dot_data = tree.export_graphviz(tr, out_file = None, feature_names = X.columns,
                                filled = True, rounded = True, special_characters = True)
graph = graphviz.Source(dot_data)
graph
```



의사결정나무

- “데이터를 분석하여 이들 사이에 존재하는 패턴을 예측 가능한 규칙들의 조합으로 나타내며, 그 모양이 나무와 같다고 하여 의사결정나무라 불린다.”
- 분류 (classification) 와 회귀 (regression) 모두 가능하다는 점에서 범주/연속형 수치 모두 예측할 수 있다는 것을 알 수 있다.
- DecisionTreeClassifier, max_depth = 3 임의 설정
- X_train, y_train fit 적합
- export_graphviz, **graphviz/sklearn** 사용해 의사결정나무 모델 가시화
- gini : 데이터 분포에서의 지니계수 (criterion)

```
In [13]: # Plot the tree
fig = plt.figure(figsize = (25, 15))
tree.plot_tree(tr.fit(X_train, y_train), feature_names = X.columns, filled = True, rounded = True, fontsize = 16);
plt.title('Decision Tree For Credit Card');
```



```
In [14]: # simplified version
r = export_text(tr, feature_names = X.columns.tolist())
print(r)
```

```

|--- PAY_0 <= 1.35
| |--- PAY_2 <= 1.36
| | |--- PAY_AMT3 <= -0.24
| | | |--- class: 0
| | | |--- PAY_AMT3 > -0.24
| | | |--- class: 0
| | |--- PAY_2 > 1.36
| | |--- PAY_5 <= 1.11
| | | |--- class: 0
| | | |--- PAY_5 > 1.11
| | | |--- class: 1
| |--- PAY_0 > 1.35
| |--- PAY_3 <= -0.28
| | |--- BILL_AMT1 <= -0.66
| | | |--- class: 0
| | | |--- BILL_AMT1 > -0.66
| | | |--- class: 1
| | |--- PAY_3 > -0.28
| | |--- PAY_6 <= 1.12
| | | |--- class: 1
| | |--- PAY_6 > 1.12
| | | |--- class: 1
  
```

```
In [15]: # Make predictions
tr_pred = tr.predict(X_test)

# CV score
tr_cv = cross_val_score(tr, X_train, y_train, cv=10).mean()
```

의사결정나무 PLOT 그리기, 트리를 단순화된 버전으로도 그려보기, 예측 값, CV(CROSS VALIDATION) 점수

Metrics for Decision tree

```
In [16]: # Accuracy: 1 is perfect prediction.
print('Accuracy: %.3f' % tr.score(X_test, y_test))

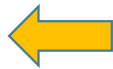
# Cross-Validation accuracy
print('Cross-validation accuracy: %.3f' % tr_cv)

# Precision
print('Precision: %.3f' % precision_score(y_test, tr_pred))

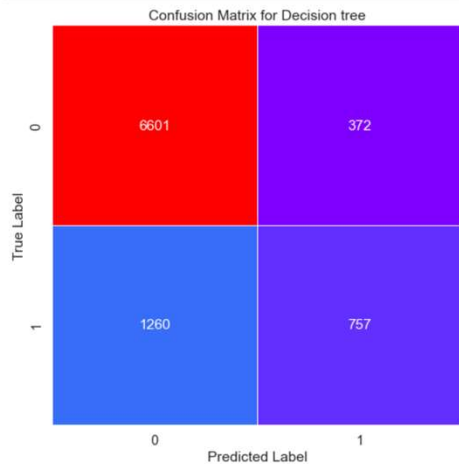
# Recall
print('Recall: %.3f' % recall_score(y_test, tr_pred))

# f1 score: best value at 1 and worst at 0.
print('F1 score: %.3f' % f1_score(y_test, tr_pred))

Accuracy: 0.818
Cross-validation accuracy: 0.822
Precision: 0.671
Recall: 0.375
F1 score: 0.481
```



```
In [17]: # Plot confusion matrix for Decision tree.
tr_matrix = confusion_matrix(y_test, tr_pred)
sns.set(font_scale = 1.3)
plt.subplots(figsize = (8, 8))
sns.heatmap(tr_matrix, annot = True, cbar = False, cmap = 'rainbow', linewidth = 0.5, fmt = "d")
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Confusion Matrix for Decision tree');
```



- Decision tree - 모델 테스트
- 척도(metric) : accuracy(정확성), cross-validation accuracy(교차검증 정확도), precision(정밀도), recall(재현도), F1 score → 확인
- y_test와 tr_pred 데이터로 confusion matrix 그려보기
- X_test 확률 예측, AUC 점수

```
In [18]: # Predict probabilities for the test data.
tr_probs = tr.predict_proba(X_test)

# Keep Probabilities of the positive class only.
tr_probs = tr_probs[:, 1]

# AUC Score
auc_tr = roc_auc_score(y_test, tr_probs)
print('AUC: %.2f' % auc_tr)

AUC: 0.73
```

AUC (Area Under the ROC Curve) : ROC curve 와 직선 사이의 면적, AUC 값의 범위는 0 ~ 1 이며 값이 클수록 예측의 정확도가 높다고 할 수 있다.

→ AUC = 0.73 도출

- Grid search for Decision tree
- Grid search 의사결정나무 척도 확인
- Confusion matrix 구현
- X_test (test data) 확률 예측, AUC 점수 (=0.75)

Confusion matrix

- : “분류 모델 성능 평가 지표”, training을 통한 prediction 성능 측정 위해 예측 value와 실제 value를 비교하기 위한 표, TP와 TN은 실제 값과 맞게 예측한 부분, FP와 FN은 실제 값과 다르게 예측 (T:TRUE, F:FALSE, P:POSITIVE, N:NEGATIVE)
→ 정확도, 정밀도, 재현도, F1 score 유도 가능!



Grid search for Decision tree

```
In [19]: parameters = {'criterion': ['gini', 'entropy'],
                    'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]}

# MLP estimator
default_tr = tree.DecisionTreeClassifier(random_state = 25)

# GridSearchCV estimator
gs_tree = GridSearchCV(default_tr, parameters, cv = 10, n_jobs = -1, verbose = 1)

# Train the GridSearchCV estimator and search for the best parameters
gs_tree.fit(X_train, y_train)

Fitting 10 folds for each of 40 candidates, totalling 400 fits

Out[19]:
* GridSearchCV
  * estimator: DecisionTreeClassifier
    * DecisionTreeClassifier

In [20]: # Make predictions with the best parameters.
gs_tree_pred = gs_tree.predict(X_test)
```

Metrics for Grid Search Decision tree

```
In [21]: # Best parameters.
print("Best Decision tree Parameters: {}".format(gs_tree.best_params_))

# Cross validation accuracy for the best parameters.
print("Cross-validation accuracy: %.3f" % gs_tree.best_score_)

# Accuracy: 1 is perfect prediction.
print("Accuracy: %.3f" % (gs_tree.score(X_test, y_test)))

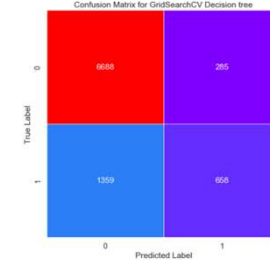
# Precision
print("Precision: %.3f" % precision_score(y_test, gs_tree_pred))

# Recall
print("Recall: %.3f" % recall_score(y_test, gs_tree_pred))

# f1 score: best value at 1 (perfect precision and recall) and worst at 0.
print("F1 score: %.3f" % f1_score(y_test, gs_tree_pred))

Best Decision tree Parameters: {'criterion': 'entropy', 'max_depth': 4}
Cross-validation accuracy: 0.822
Accuracy: 0.817
Precision: 0.698
Recall: 0.326
F1 score: 0.445
```

```
In [22]: # Plot confusion matrix for Decision tree
gs_tr_matrix = confusion_matrix(y_test, gs_tree_pred)
sns.set(font_scale = 1.2)
plt.subplots(figsize = (5, 8))
sns.heatmap(gs_tr_matrix, annot = True, cbar = False, cmap = 'rainbow', linewidth = 0.5, tet = 'td')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for GridSearchCV Decision tree')
```



```
In [23]: # Predict probabilities for the test data.
gs_tree_probs = gs_tree.predict_proba(X_test)

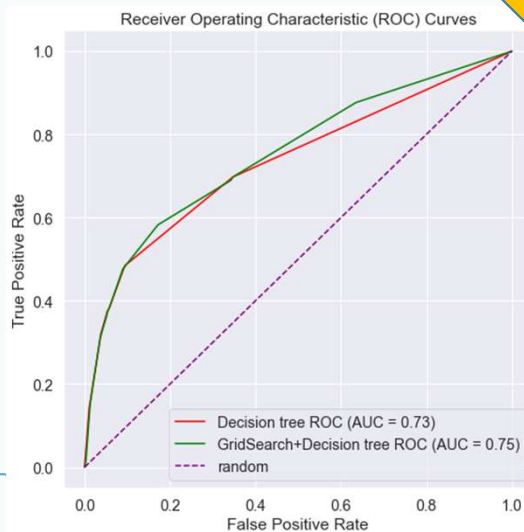
# Keep Probabilities of the positive class only.
gs_tree_probs = gs_tree_probs[:, 1]

# AUC Score
gs_tree_auc = roc_auc_score(y_test, gs_tree_probs)
print("AUC: %.2f" % gs_tree_auc)

AUC: 0.75
```

```
In [24]: # Get the ROC Curves.
gs_tr_fpr, gs_tr_tpr, gs_tr_thresholds = roc_curve(y_test, gs_tree_probs)
tr_fpr, tr_tpr, tr_thresholds = roc_curve(y_test, tr_probs)

# Plot the ROC curves.
plt.figure(figsize=(8,8))
plt.plot(tr_fpr, tr_tpr, color = 'red', label = 'Decision tree ROC (AUC = %0.2f) % auc_tr)
plt.plot(gs_tr_fpr, gs_tr_tpr, color = 'green', label = 'GridSearch+Decision tree ROC (AUC = %0.2f) % gs_tree_auc)
plt.plot([0, 1], [0, 1], color = 'purple', linestyle = '--', label = 'random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curves')
plt.legend()
plt.show()
```



→ ROC curve for
 Decision tree
 VS
 Grid Search + Decision tree
 VS
 random

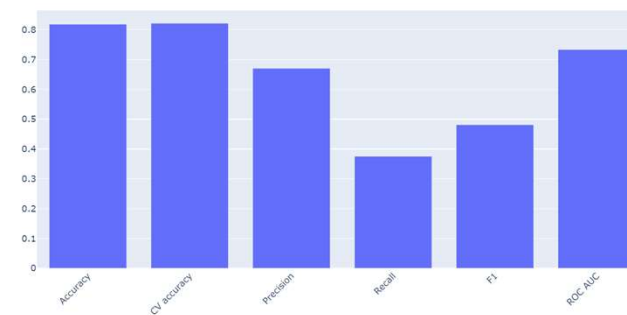
Results

```
In [25]: metrics=['Accuracy', 'CV accuracy', 'Precision', 'Recall', 'F1', 'ROC AUC']

fig = go.Figure(data=[go.Bar(name='Decision tree', x=metrics,
y=[tr_score(x_test, y_test), tr_cv.precision_score(y_test, tr_pred),
recall_score(y_test, tr_pred), f1_score(y_test, tr_pred), auc_tr)])])

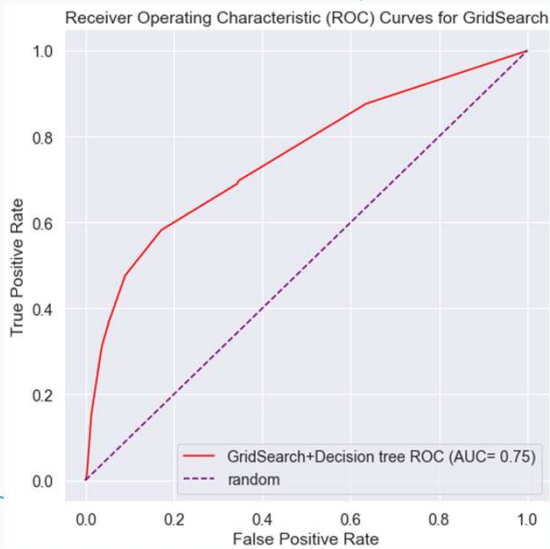
fig.update_layout(title_text='Metrics for each', bargroup='group', xaxis_tickangle=-45, bargroupgap=0.05)
fig.show()
```

Metrics for each



- ROC (Receiver Operating Characteristic) curve 그려보기
 → 직관적으로 파악 가능
- 의사결정나무 척도 5개 + ROC AUC 대해 hist 구현


```
In [26]: # Plot the ROC curve.
plt.figure(figsize=(8,8))
plt.plot(gs_tr_fpr, gs_tr_tpr, color = 'red', label = 'GridSearch+Decision tree ROC (AUC= %.2f)' % gs_tree_auc)
plt.plot([0, 1], [0, 1], color = 'purple', linestyle = '--', label = 'random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curves for GridSearch')
plt.legend()
plt.show()
```



- GridSearch + Decision tree ROC curve 그려보기
- 두 종류 모델 척도 → DataFrame 으로 나타내기
- 척도 + ROC AUC 마다 두 모델 비교 가능 ↑ ↓

```
In [27]: d={
    '': ['Decision Tree', 'GridSearchCV + Decision Tree'],
    'Accuracy': [tr.score(X_test, y_test), gs_tree.score(X_test, y_test)],
    'CV Accuracy': [tr_cv, gs_tree.best_score_],
    'Precision': [precision_score(y_test, tr_pred), precision_score(y_test, gs_tree_pred)],
    'Recall': [recall_score(y_test, tr_pred), recall_score(y_test, gs_tree_pred)],
    'F1': [f1_score(y_test, tr_pred), f1_score(y_test, gs_tree_pred)],
    'ROC AUC': [auc_tr, gs_tree_auc]
}

results = pd.DataFrame(data = d).round(3).set_index('')
results
```

Out [27]:



	Accuracy	CV Accuracy	Precision	Recall	F1	ROC AUC
Decision Tree	0.818	0.822	0.671	0.375	0.481	0.733
GridSearchCV + Decision Tree	0.817	0.822	0.698	0.326	0.445	0.752

+) 교차검증, 가지치기 수행 및 분석

Q1. 의사결정나무가 결측치에 예민한가?

A1. 의사결정나무(Decision Tree)는 결측치에 민감하지 않은 모델 중 하나

☞ 결측치 있는 데이터를 처리하는 데 일반적으로 많이 사용되는 방법 중 하나는 “대체 (Imputation)”

: 결측치가 있는 변수를 다른 변수의 값이나 전체 데이터의 통계치 등으로 대체하는 것

→ 결측치 대체하면 다양한 머신러닝 모델에서 결측치에 대한 예측을 수행할 수 있다!

BUT 결측치가 있는 변수가 많을 경우 → 대체를 수행하는 것이 복잡하고 시간이 오래 걸리는 작업이 될 수 있다.

=> 따라서, 결측치가 많은 데이터의 경우에는 결측치를 대체 X, 해당 변수를 **제거**하는 것이 더 나은 결과를 얻을 수 있는 경우도 존재

따라서, 결측치가 있는 데이터를 다룰 때는 의사결정나무가 결측치에 예민한 모델은 아니지만, 결측치를 어떻게 처리할 것인지에 따라 모델의 성능이 달라질 수 있다는 점을 염두에 두어야 한다.

→ 이를 위해서는 데이터의 결측치를 제거하거나 대체하는 방법에 대해 신중하게 고려

추가 코드!!

1. 실험환경 구성

```
In [36]: # 라이브러리 로드
import pandas as pd

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split # 학습, 테스트셋 구분
from sklearn.tree import export_graphviz # tree 시각화를 위한

# export_graphviz : 의사결정나무에 대한 graphviz dot data 생성하는 함수
import graphviz # tree 시각화
import sklearn.metrics as mt # 성능지표를 계산하기 위해 import
from sklearn.model_selection import cross_val_score, cross_validate # 교차검증

import warnings
warnings.filterwarnings('ignore')
```

2. 교차검증 Cross Validation

```
In [37]: dt_clf = DecisionTreeClassifier(random_state=5764)
dt_clf.fit(X_train, y_train) # 학습

# 각 폴드의 스코어
scores = cross_val_score(dt_clf, X, y, cv = 5)

print('Averaged results of cross validation: ', scores.mean())

Averaged results of cross validation: 0.7238444852327716
```

```
In [38]: pd.DataFrame(cross_validate(dt_clf, X, y, cv=5))
```

```
Out[38]:
```

	fit_time	score_time	test_score
0	0.423647	0.001998	0.708493
1	0.421939	0.003039	0.713165
2	0.426843	0.003009	0.724178
3	0.407325	0.002991	0.739696
4	0.392098	0.002503	0.733689

```
In [39]: # test set에 대한 스코어(정확도)
dt_clf.score(X_test, y_test)
```

```
Out[39]: 0.7253615127919911
```

교차검증 수행 (정확도) → AVERAGED RESULT, TEST SET SCORE?

3. 가지치기 수행 Pruning

```
In [32]: pruned_dt_clf = DecisionTreeClassifier(max_depth = 2, random_state = 156) # max_depth = 2로 제한
pruned_dt_clf.fit(X_train, y_train)

print("Accuracy of training set: {:.3f}".format(pruned_dt_clf.score(X_train, y_train)))
print("Accuracy of test set: {:.3f}".format(pruned_dt_clf.score(X_test, y_test)))

Accuracy of training set: 0.821
Accuracy of test set: 0.818
```

- 가지치기 수행 전보다 test set에 대한 정확도가 향상됨을 확인할 수 있다! (0.821 -> 0.818)

Q2. 가지치기의 정도?

A2. 의사결정나무에서 가지치기의 정도를 결정하는 것은 중요한 문제 중 하나

- 가지치기를 너무 많이 하면 과소적합(underfitting)의 문제가, 반대로 가지치기를 너무 적게 하면 과대적합(overfitting)의 문제가 발생할 수 있다.

☞ 이를 해결하기 위해, 보통은 검증(validation) 데이터를 사용하여 가지치기의 정도를 결정

- 검증 데이터를 사용하여 모델을 평가 → 가지치기를 다양한 정도로 적용하여 각각의 경우의 검증 데이터에 대한 성능 측정 → 그리고 검증 데이터에 대한 성능이 가장 좋은 가지치기의 정도를 최종 모델의 가지치기 정도로 결정

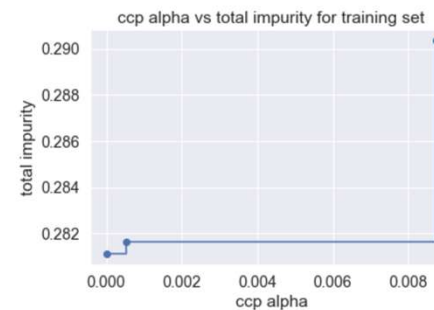
3-1. 비용복잡도 가지치기 Cost-complexity Pruning

```
In [41]: path = pruned_dt_clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
print(ccp_alphas, impurities)

fig, ax = plt.subplots()
ax.plot(ccp_alphas[:-1], impurities[:-1], marker="o", drawstyle="steps-post")
ax.set_xlabel("ccp alpha")
ax.set_ylabel("total impurity")
ax.set_title("ccp alpha vs total impurity for training set")

[0.          0.00052225 0.00874155 0.05275222] [0.28110386 0.28162612 0.29036767 0.34311889]

Out[41]: Text(0.5, 1.0, 'ccp alpha vs total impurity for training set')
```



가지치기 적용 후 → 의사결정나무의 **정확도 향상!** + 비용복잡도 가지치기 CC(T) 수행

```
In [34]: import pandas as pd
import matplotlib.pyplot as plt # package for plotting
from sklearn.tree import DecisionTreeClassifier, plot_tree
plt.rcParams['axes.unicode_minus'] = False # 마이너스 부호 깨짐 현상

In [35]: data = pd.read_csv("default of credit card clients.csv", index_col=0)
features = data.columns
print(features)

Index(['LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0', 'PAY_2',
      'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
      'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
      'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6',
      'default payment next month'],
      dtype='object')
```

Q3. 학습데이터에 과적합 문제가 있는가? (제시 코드/결과 참조)

A3-1. 훈련 데이터와 검증 데이터의 성능 비교

- 훈련 데이터에 대해 과적합이 발생할 경우, 훈련 데이터에 대한 예측 정확도는 높지만, 검증 데이터에 대한 예측 정확도는 낮아진다.

A3-2. 교차검증

A3-3. 특성 중요도

- 특정 변수의 중요도가 높다 = 해당 변수가 예측 결과에 영향을 많이 미친다. 따라서 모델에서 특정 변수의 중요도가 높을 경우, 해당 변수가 과적합 문제를 일으키고 있을 수 있다.

```
In [36]: from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# 의사결정나무 모델 객체 생성
tree = DecisionTreeClassifier()

# 하이퍼파라미터 그리드 설정
param_grid = {'max_depth': range(1, 11), 'min_samples_split': range(2, 11)}

# GridSearchCV를 사용하여 가지치기 조정
grid = GridSearchCV(tree, param_grid = param_grid, cv = 5)
grid.fit(X_train, y_train)

# 최적의 하이퍼파라미터와 검증 데이터에서의 최고 정확도 출력
print("Best parameter: ", grid.best_params_)
print("Best cross-validation score: {:.2f}".format(grid.best_score_))
```

```
Best parameter: {'max_depth': 4, 'min_samples_split': 2}
Best cross-validation score: 0.82
```

- max_depth와 min_samples_split이라는 두 개의 하이퍼파라미터를 그리드로 설정
- 5-fold 교차 검증(cross-validation)을 수행하여 최적의 하이퍼파라미터와 검증 데이터에서의 최고 정확도를 출력
- 이를 통해 가지치기의 정도를 결정 가능 => max-depth = 4, min_samples_split = 2, best cv score = 0.82

가지치기의 정도 결정 → **MAX_DEPTH = 4**, MIN_SAMPLES_SPLIT = 2, BEST CV SCORE = 0.82

최적의 하이퍼파라미터로 가지치기 재수행

```
In [39]: from sklearn import tree

# Initialize a decision tree estimator
tr1 = tree.DecisionTreeClassifier(max_depth = 4, criterion = 'gini', random_state = 25)

# Train the estimator
tr1.fit(X_train, y_train)
```

```
Out[39]: DecisionTreeClassifier(max_depth=4, random_state=25)
```

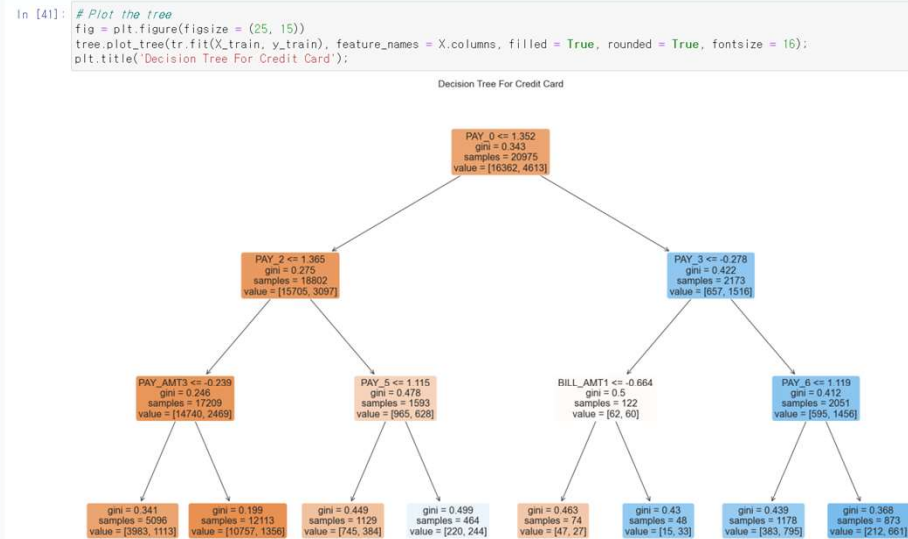
```
In [40]: # Plot the tree.
dot_data = tree.export_graphviz(tr, out_file = None, feature_names = X.columns,
                                filled = True, rounded = True, special_characters = True)
graph = graphviz.Source(dot_data)
graph
```

```
In [42]: # simplified version
r = export_text(tr1, feature_names = X.columns.tolist())
print(r)
```

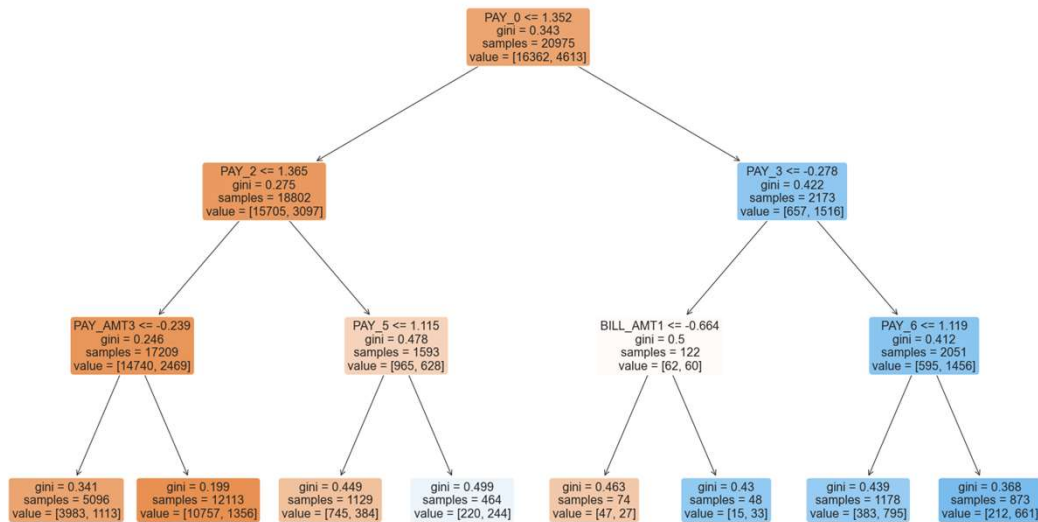
```
|--- PAY_0 <= 1.35
|   |--- PAY_2 <= 1.36
|   |   |--- PAY_AMT3 <= -0.24
|   |   |   |--- BILL_AMT1 <= -0.69
|   |   |   |   |--- class: 0
|   |   |   |   |--- BILL_AMT1 > -0.69
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- PAY_AMT3 > -0.24
|   |   |   |   |   |--- PAY_4 <= 0.62
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- PAY_4 > 0.62
|   |   |   |   |   |   |   |--- class: 0
|   |   |--- PAY_2 > 1.36
|   |   |   |--- PAY_5 <= 1.11
|   |   |   |   |--- LIMIT_BAL <= 0.13
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- LIMIT_BAL > 0.13
|   |   |   |   |   |--- class: 0
|   |   |   |--- PAY_5 > 1.11
|   |   |   |   |--- PAY_AMT5 <= -0.31
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- PAY_AMT5 > -0.31
|   |   |   |   |   |--- class: 0
|   |--- PAY_0 > 1.35
|   |   |--- PAY_3 <= -0.28
|   |   |   |--- BILL_AMT1 <= -0.66
|   |   |   |   |--- BILL_AMT2 <= -0.68
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- BILL_AMT2 > -0.68
|   |   |   |   |   |--- class: 0
|   |   |   |--- BILL_AMT1 > -0.66
|   |   |   |   |--- PAY_AMT5 <= 0.17
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- PAY_AMT5 > 0.17
|   |   |   |   |   |--- class: 0
|   |   |--- PAY_3 > -0.28
|   |   |   |--- PAY_6 <= 1.12
|   |   |   |   |--- EDUCATION <= 3.35
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- EDUCATION > 3.35
|   |   |   |   |   |--- class: 0
|   |   |   |--- PAY_6 > 1.12
|   |   |   |   |--- PAY_AMT3 <= 0.53
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- PAY_AMT3 > 0.53
|   |   |   |   |   |--- class: 0
```



- 앞에서 구한 최적의 파라미터로 가지치기에 적용해 의사결정나무 모델 가시화 : $\text{max_depth} = 4$
- 단순화된 버전으로도 그려보면, 전과 다른 결과가 나오는 것을 확인 가능



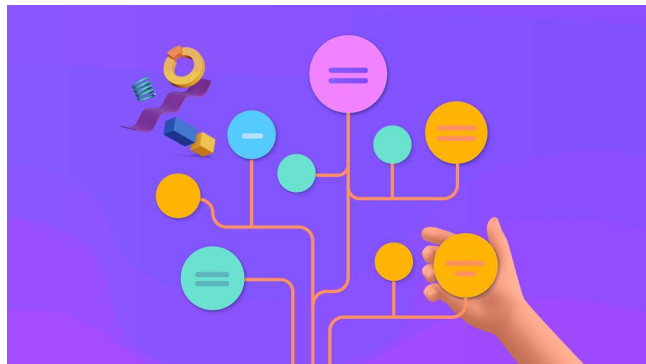
Decision Tree For Credit Card



+) 의사결정나무의 특징

- “주어진 입력값에 대하여 출력값을 예측하는 모형”
- → 나무 형태의 그래프로 표현
- 예측력은 다른 지도학습 기법들에 비해 대체로 떨어지지만, 해석력 좋음
- 1) 분류 나무(classification tree) : 출력 변수가 범주형
- 2) 회귀 나무(regression tree) : 출력 변수가 연속형
- ① 성장 (tree growing): 최대 크기의 나무 모형 형성
- → 각 마디에서 적절한 최적의 분리규칙을 찾아서 나무를 성장시키는 과정으로서 적절한 정지규칙을 만족하면 중단
- ② 가지치기 (pruning): 최대 크기 나무모형에서 불필요한 가지를 제거하여 부분 나무모형 (subtrees)의 집합을 탐색
- ③ 최적 나무모형 선택: 가지치기의 결과인 나무모형의 집합에서 최적의 모형을 선택
- → 검증오차가 가장 작은 의사결정나무를 평가
- ④ 해석 및 예측: 구축된 나무모형을 해석하고 예측모형을 설정한 후 예측에 적용

의사결정나무의 장단점



• 의사결정나무의 장점

- ① 의사결정나무 (특히 CART : “각각 종속 변수가 범주 or 숫자인지 여부에 따라 분류/회귀 나무 중 하나를 생성하는 비모수적 결정 나무 학습법”)는 IF-THEN 형식으로, 이해하기 쉬운 규칙
- ② 연속형 변수와 범주형 변수 **모두** 취급 가능
- ③ 모형에 대한 가정 (예 : 선형회귀의 선형성, 등분산성 등)이 필요 **없는** 비모수적 방법
- ④ 가장 설명력이 있는 변수에 대하여 최초로 **분리**가 일어난다

• 의사결정나무의 단점

- ① 출력변수가 연속형인 회귀모형에서는 → 예측력 저하 ↓
- ② 일반적으로 복잡한 나무 모형은 예측력 저하 ↓ 되고 해석 어려움
- ③ 상황에 따라 계산량이 많을 수도 있음, 베이지 분류경계가 사각형이 아닌 경우에 좋지 않은 결과 도출
- ④ 자료에 변화가 있는 경우에 전혀 다른 결과를 (즉, 분산이 매우 큰) 줄 수도 있는 불안정한 방법 - ex) 배깅(bagging)과 같은 앙상블 알고리즘