병원 개/폐업 분류 예측 경진대회

Machine Learning

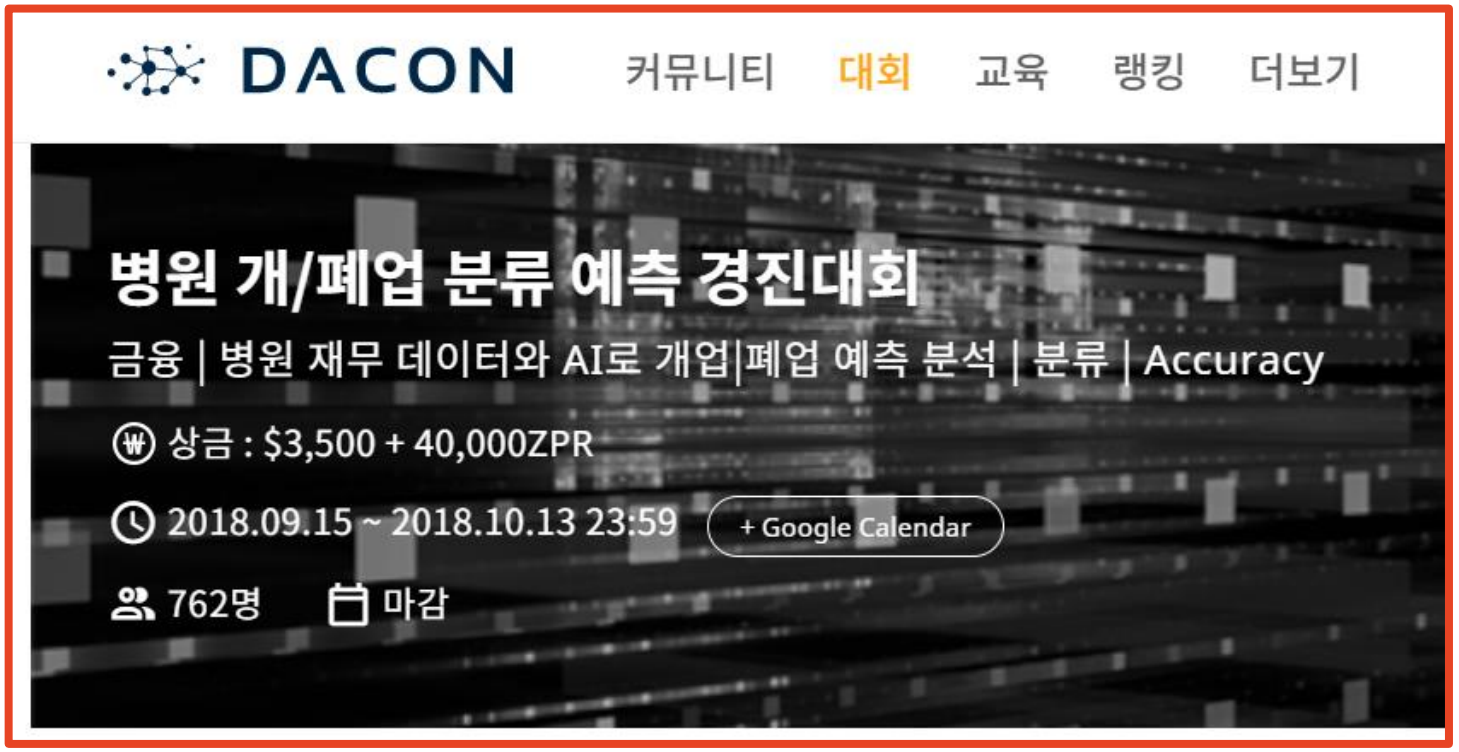
15기 김지호
15기 우명진
15기 장수혁
15기 조우영

목차

01

대회 개요

-대회 목적



상환 기간 동안의 경영 여부를 예측하여
폐업하는 병원을 예측하기

-> 폐업 예측을 기반으로 대출 여부를 결정

평가 척도: Accuracy

01

데이터 개요

변수 설명

301 rows x 58 columns

<Target 변수>

```
train['OC'].unique()
```

```
array(['open', ' close'], dtype=object)
```

- inst_id - 각 파일에서의 병원 고유 번호
- OC – 영업/폐업 분류, 2018년 폐업은 2017년 폐업으로 간주함
- sido – 병원의 광역 지역 정보
- sgg – 병원의 시군구 자료
- openDate – 병원 설립일

병원 규모

- bedCount - 병원이 갖추고 있는 병상의 수
- instkind – 병원, 의원, 요양병원, 한의원, 종합병원 등 병원의 종류
  - 종합병원 : 입원환자 100명 이상 수용 가능
  - 병원 : 입원 환자 30명 이상 100명 미만 수용 가능
  - 의원 : 입원 환자 30명 이하 수용 가능
  - 한방 병원(한의원) : 침술과 한약으로 치료하는 의료 기관.

- 수익
  - revenue1 – 매출액
  - noi1 – 영업외수익
- 비용
  - salescost1 – 매출원가
  - sga1 - 판매비와 관리비
  - salary1 – 급여
  - noe1 – 영업외비용
  - Interest1 – 이자비용
  - ctax1 – 법인세비용
- Profit1 – 당기순이익
- 자산
  - liquidAsset1 – 유동자산
    - quickAsset1 – 당좌자산
      - receivableS1 - 미수금(단기)
    - inventoryAsset1 – 재고자산
- nonCAsset1 – 비유동자산
  - tanAsset1 – 유형자산
  - OnonCAsset1 - 기타 비유동자산
    - receivableL1 – 장기미수금

- 부채
  - debt1 – 부채총계
  - liquidLiabilities1 – 유동부채
    - shortLoan1 – 단기차입금
  - NCLiabilities1 – 비유동부채
    - longLoan1 – 장기차입금
- 자본
  - netAsset1 – 순자산총계
  - surplus1 – 이익잉여금 : 순이익 누적금액

인적 자원 --> 1. 의료의 질 2. 이직률

- employee1 – 고용한 총 직원의 수, 2017
- employee2 – 고용한 총 직원의 수, 2016
- ownerChange – 대표자의 변동

02 Train 데이터 전처리

1. 불필요한 정보 처리

#병원 고유 번호, 시군구 자료 drop

```
train.drop(["inst_id","sgg"],axis=1,inplace=True)
```

#target 변수 'OC' 수정

```
train["OC"].unique()
```

```
array(['open', ' close'], dtype=object)
```

```
#OC열 close 앞 띄어쓰기 제거
train["OC"] = train["OC"].str.replace(" ","")
```

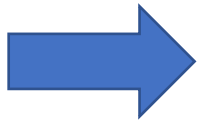
```
train["OC"].unique()
```

```
array(['open', 'close'], dtype=object)
```

#openDate를 경력으로 바꾸기

```
train['career'] = train['openDate'].astype('str') #int인 openDate를 str로 바꿔주기
train['career'] = pd.to_datetime(train['career']) #str를 날짜 형식으로 바꿔주기
train['career'] = pd.to_datetime('20180101') - train['career'] #20181801010에서 뺀 기간으로 바꿔주기
```

career
3657 days
369 days
6349 days



```
train["career"]
```

```
0       3657
1       7580
2        369
3       6349
4       4505
       ...
296     4707
297     1293
298    12599
299     4018
300     5920
```

02 Train 데이터 전처리

1. 불필요한 정보 처리

회계 정보가 중복되는 병원

```
train1[train1.duplicated()].shape #총 14개의 행 중복
```

```
(14, 48)
```

```
train.drop_duplicates(subset=train1.columns,keep="first",inplace=True) #14개 행 모두 삭제
```

개원일과 회계정보가 불일치하는 병원

2017 개원인데, 2016년 data가 존재?

```
train[train.career<370]
```

경력일이 365 이하 = 2017년 개원한 병원

	revenue1	salescost1	sga1	salary1	noi1	noe1	interest1
	1.004522e+09	5.154837e+08	4.672197e+08	2.964023e+08	76156.0	30000.0	0.0
	1.033288e+10	4.075620e+09	1.000945e+09	1.096381e+09	71250013.0	282249363.0	282249363.0
	5.236406e+08	2.306773e+07	8.155775e+08	5.644136e+08	7534205.0	52630597.0	52630597.0

2016년 회계 정보

02 Train 데이터 전처리

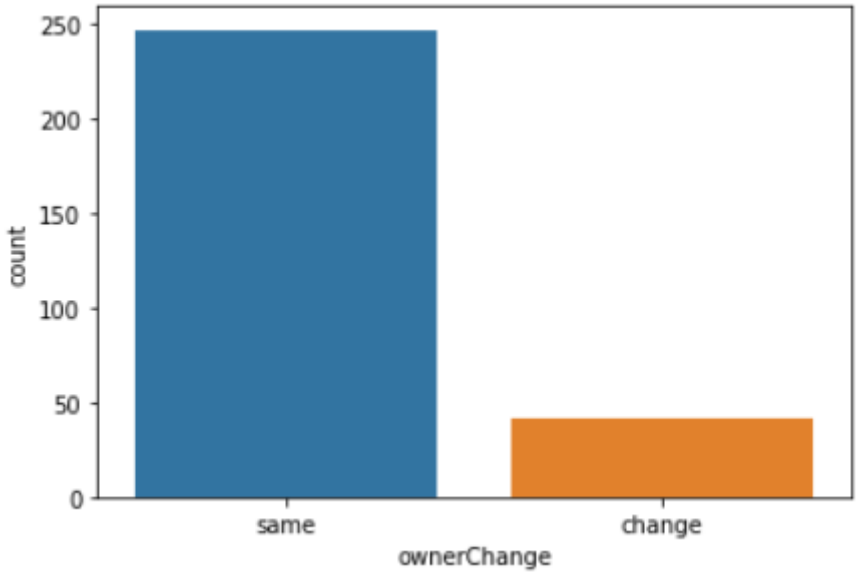
2. 결측치 처리

OC	0	revenue2	8
sido	0	salescost2	8
bedCount	5	sga2	8
instkind	1	salary2	8
revenue1	8	noi2	8
salescost1	8	noe2	8
sga1	8	interest2	8
salary1	8	ctax2	8
noi1	8	profit2	8
noe1	8	liquidAsset2	8
interest1	8	quickAsset2	8
ctax1	8	receivableS2	8
profit1	8	inventoryAsset2	8
liquidAsset1	8	nonCAsset2	8
quickAsset1	8	tanAsset2	8
receivableS1	8	OnonCAsset2	8
inventoryAsset1	8	receivableL2	8
nonCAsset1	8	debt2	8
tanAsset1	8	liquidLiabilities2	8
OnonCAsset1	8	shortLoan2	8
receivableL1	8	NCLiabilities2	8
debt1	8	longLoan2	8
liquidLiabilities1	8	netAsset2	8
shortLoan1	8	surplus2	8
NCLiabilities1	8	employee1	10
longLoan1	8	employee2	13
netAsset1	8	ownerChange	12
surplus1	8		

#ownerChange

```
train['ownerChange'].value_counts()
```

```
same      247
change     42
```




```
#ownerChange 결측치 최빈값으로 대체
print("ownerChange 최빈값 : ", train['ownerChange'].mode())
most_value1 = train['ownerChange'].value_counts(dropna = True).idxmax()
train['ownerChange'].fillna(most_value1, inplace = True)
```

```
ownerChange 최빈값 :  0    same
dtype: object
```

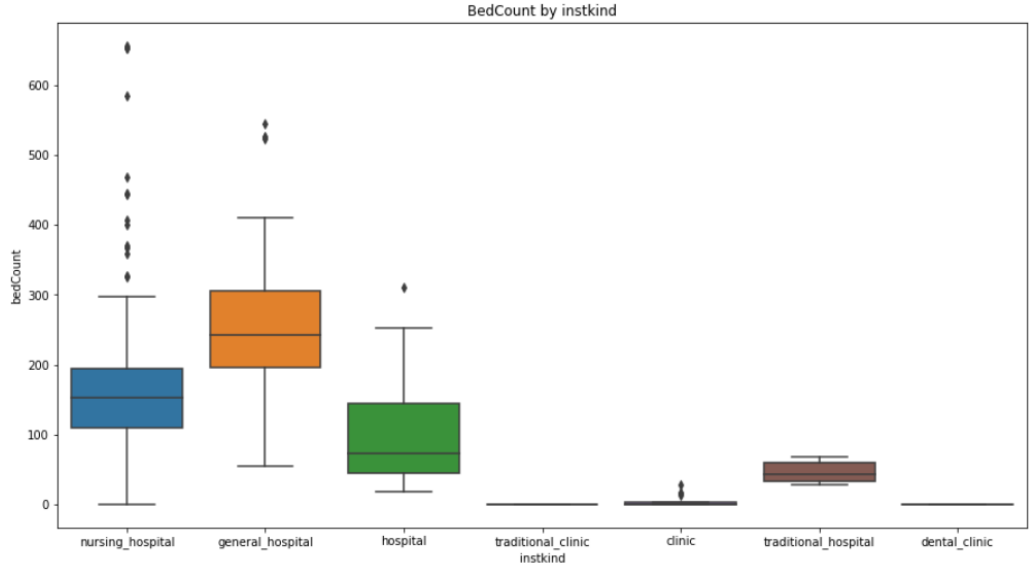
02

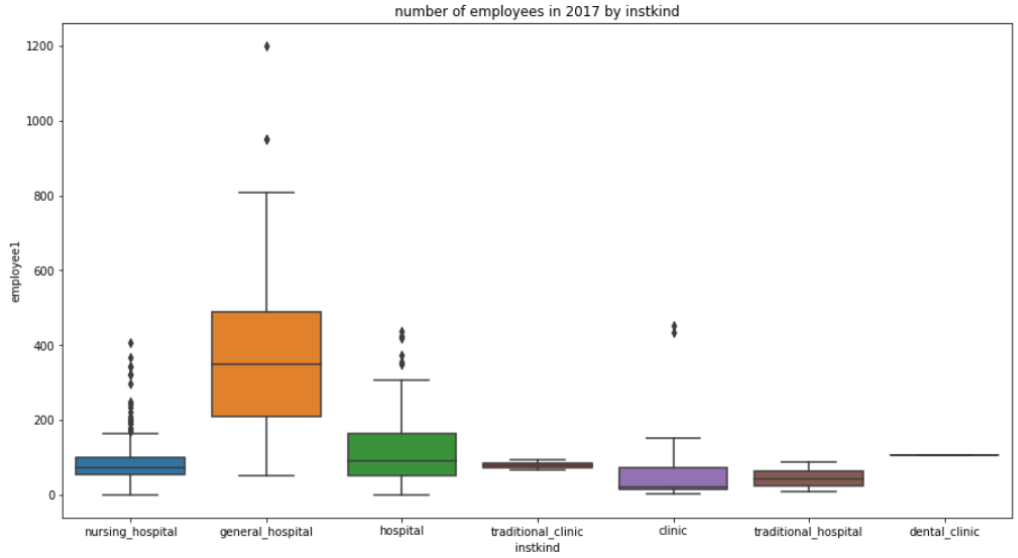
Train 데이터 전처리

2. 결측치 처리

#bedCount #employee

	bedCount	instkind
71	NaN	traditional_hospital
193	NaN	NaN
297	NaN	hospital
298	NaN	hospital
300	NaN	traditional_hospital



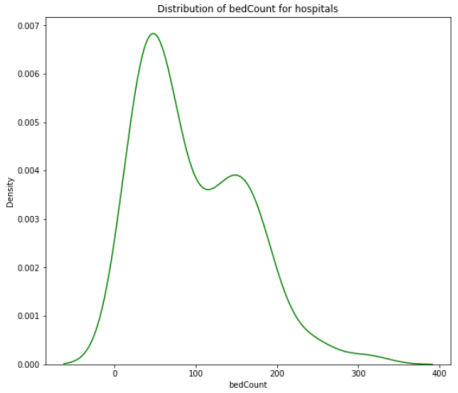


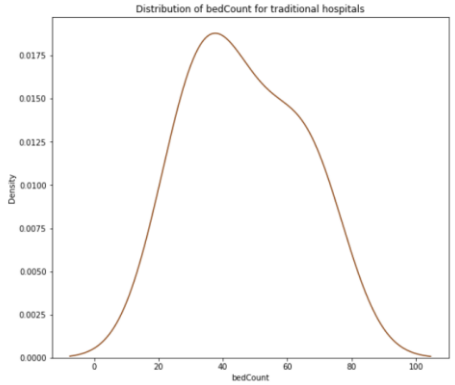
02

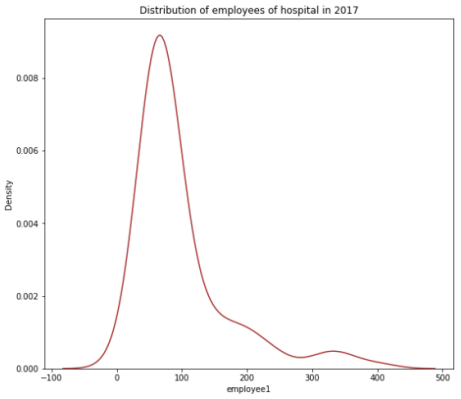
Train 데이터 전처리

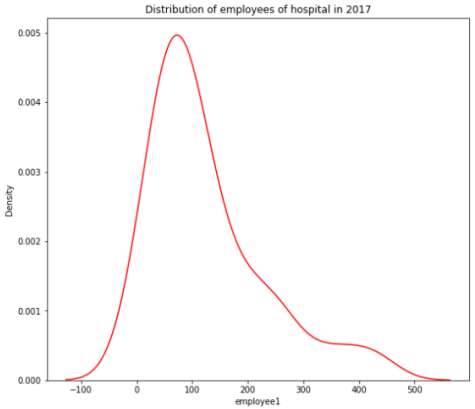
2. 결측치 처리

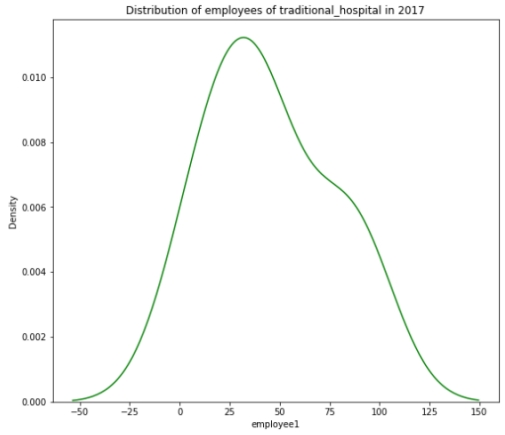
#bedCount #employee












```
train.loc[train['instkind'] == 'hospital', 'bedCount'] = train.loc[train['instkind'] == 'hospital', 'bedCount'].fillna(train.loc[train['instkind'] == 'hospital', 'bedCount'].median())
train.loc[train['instkind'] == 'traditional_hospital', 'bedCount'] = train.loc[train['instkind'] == 'traditional_hospital', 'bedCount'].fillna(train.loc[train['instkind'] == 'traditional_hospital', 'bedCount'].median())
```

```
train.loc[train['instkind'] == 'traditional_hospital', 'employee1'] = train.loc[train['instkind'] == 'traditional_hospital', 'employee1'].fillna(train.loc[train['instkind'] == 'traditional_hospital', 'employee1'].median())
train.loc[train['instkind'] == 'hospital', 'employee1'] = train.loc[train['instkind'] == 'hospital', 'employee1'].fillna(train.loc[train['instkind'] == 'hospital', 'employee1'].median())
train.loc[train['instkind'] == 'nursing_hospital', 'employee1'] = train.loc[train['instkind'] == 'nursing_hospital', 'employee1'].fillna(train.loc[train['instkind'] == 'nursing_hospital', 'employee1'].median())
```

02 Train 데이터 전처리

2. 결측치 처리

#bedCount와 instkind 모두 결측치인 행

	employee1	employee2
193	15.0	15.0

Instkind를 'clinic'으로 채워줌

```
train.groupby('instkind').describe()[["employee1","employee2"]]
```

	employee1								employee2							
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max
instkind																
clinic	18.0	81.222222	136.887053	4.0	16.00	21.0	65.50	454.0	18.0	76.666667	128.167170	6.0	15.00	23.0	63.25	425.0
dental_clinic	1.0	107.000000	NaN	107.0	107.00	107.0	107.00	107.0	1.0	109.000000	NaN	109.0	109.00	109.0	109.00	109.0
general_hospital	37.0	406.216216	259.850078	53.0	211.00	350.0	489.00	1200.0	37.0	384.364865	249.699671	53.0	213.00	311.5	489.00	1200.0
hospital	88.0	123.875000	101.437690	1.0	58.25	90.0	162.50	436.0	88.0	114.806818	89.008634	1.0	54.50	89.5	146.00	418.0
nursing_hospital	144.0	97.305556	73.385206	0.0	54.00	73.0	100.00	406.0	144.0	93.062500	68.100627	0.0	56.75	74.0	95.00	454.0
traditional_clinic	4.0	79.500000	10.630146	67.0	76.00	79.0	82.50	93.0	4.0	75.500000	13.076697	67.0	69.25	70.0	76.25	95.0
traditional_hospital	8.0	44.625000	28.500313	9.0	30.00	42.0	52.75	87.0	8.0	40.000000	27.181927	10.0	27.75	33.0	44.75	88.0

02

Train 데이터 전처리

3. 0 처리

Debt

총 부채 = 유동 부채 + 비유동 부채

```
train[train["debt1"]==0.0][["debt1","liquidLiabilities1","NCLiabilities1"]]
```

	debt1	liquidLiabilities1	NCLiabilities1
5	0.0	1.849938e+10	0.0
8	0.0	1.155913e+1	21180790.0
18	0.0	1.316055e+10	0.0
35	0.0	0.000000e+00	0.0

Net Asset

순자산총계 = 유동자산 + 비유동자산 - 부채총계

```
train[train["netAsset1"]==0.0][["netAsset1","liquidAsset1","NCLiabilities1","debt1"]]
```

	netAsset1	liquidAsset1	NCLiabilities1	debt1
5	0.0	5.635105e+09	0.0	0.000000e+00
8	0.0	4.134273e+	21180790.	0.000000e+00
18	0.0	8.194001e+08	0.0	0.000000e+00
35	0.0	0.000000e+00	0.0	0.000000e+00

Noe

영업 외 비용 = 이자비용 + 기타..

Data에 없는

	noe1	interest1
52	0.0	169068842.0
94	0.0	0.0
140	0.0	113003550.0
186	0.0	0.0

02

Train 데이터 전처리

3. 0 처리

Salary

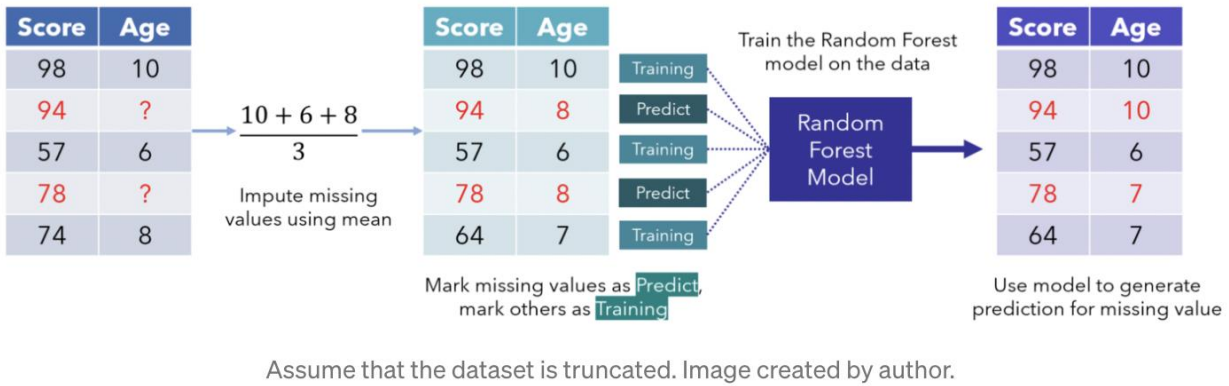
employee가 있는데 salary가 0?

Revenue

revenue가 0?

정보 누락을 의미하는 0 > imputer로 채움

〈Miss Forest imputer〉



Assume that the dataset is truncated. Image created by author.

```
imputer = MissForest(random_state=42)

miss=X_num
miss_imputed = imputer.fit_transform(miss)
```

```
Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
Iteration: 6
Iteration: 7
Iteration: 8
Iteration: 9
```

02

Train 데이터 전처리

4. 새로운 변수 생성

#유출직원수 = 2016년 직원 수(employee1) – 2017년 직원 수(employee2)

```
train["outflow"] = train["employee1"] - train["employee2"]
```

#자산총계 = 유동자산(liquidAsset) + 비유동자산(nonCAsset)

```
X["Asset1"] = X["liquidAsset1"] + X["nonCAsset1"]
X["Asset2"] = X["liquidAsset2"] + X["nonCAsset2"]
```

#매출총이익 = 매출액(revenue) – 매출원가(salecost)

```
X['grossprofit1'] = X['revenue1'] - X['salescost1']
X['grossprofit2'] = X['revenue2'] - X['salescost2']
```

#영업이익 = 매출총이익 – 판매 및 관리비(sga)

```
X['operating_profit1'] = X['revenue1'] - X['salescost1'] - X['sga1']
X['operating_profit2'] = X['revenue2'] - X['salescost2'] - X['sga2']
```

#매출액 대비 영업이익률 = 영업이익 / 매출액(revenue)

```
X['opr1'] = X['operating_profit1']/X['revenue1']
X['opr2'] = X['operating_profit2']/X['revenue2']
```

02 Test 데이터 전처리

#employee 쉼표 없애기, 자료형 바꾸기

1,637	1,547
1,816	1,104

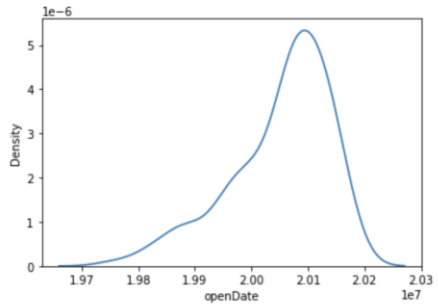
```
[ ]  #test employee열 쉼표없애기
     test['employee1'] = test['employee1'].str.replace(',', '')
     test['employee2'] = test['employee2'].str.replace(',', '')
```

```
[ ]  #test datatype obj >> float로 바꿔주기
     test['employee1'] = test['employee1'].astype(float)
     test['employee2'] = test['employee2'].astype(float)
```

openDate 결측치 → 최빈값 대체 → 'career' 로 변경

```
[ ]  sns.kdeplot(test["openDate"])
     #오른쪽으로 치우쳐있는 모습 (왜도 <0 )
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f86d769f750>
```




```
[ ]  test["openDate"].mode()
```

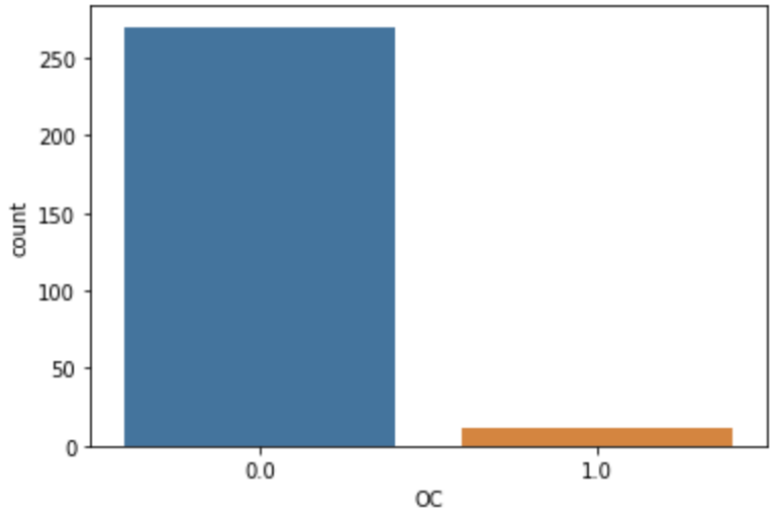
```
0    20030102.0
1    20100226.0
2    20110901.0
3    20121030.0
4    20161229.0
5    20170531.0
dtype: float64
```

```
[ ]  test["openDate"].fillna(test["openDate"].mode()[1],inplace=True)
```

02

Test 데이터 전처리

불균형 클래스 문제



SMOTE, ADASYN, SMOTE-Tomek

ADASYN

SMOTE를 개선한 방법. 샘플링 개수를 데이터 위치에 따라 다르게 설정
소수 클래스 주변의 다수 클래스 수에 따라 유동적으로 오버샘플링 할 데이터 개수를 생성한다.

```
[ ]  from imblearn.over_sampling import ADASYN

     X_resampled, y_resampled = ADASYN(random_state=42).fit_resample(X, y)
```

```
[ ]  X,y = X_resampled, y_resampled
```

```
[ ]  y.value_counts()
```

Train / test split

```
[ ]  from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42, stratify=y)
```

Scaling

1. Standard Scaler: Standardization:

$$z = \frac{x - \mu}{\sigma}$$

2. Minmax Scaler:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

3. Robust Scaler:

$$X_{\text{scale}} = \frac{x_i - x_{\text{med}}}{x_{75} - x_{25}}$$

03

모델 선택

성능 평가 척도

1. RepeatedStratifiedKFold = RepeatedKFold + StratifiedKFold

   데이터의 폴드를 지정한 횟수만큼 반복해서 나눔(Repeated) + Target Class 비율을 고려(Stratified)

2. Cross_val_score: 교차 검증을 위한 함수

   X_train, y_train을 validation set으로도 활용할 수 있음

```
[ ]
1 from sklearn.model_selection import cross_val_score
2 from sklearn.model_selection import RepeatedStratifiedKFold
3
4 def scoring(model,feature):
5     rskfold = RepeatedStratifiedKFold(n_splits=5, n_repeats=5, random_state=42)  #5개로 5번 split
6     #클래스의 표본비율을 전체집합과 동일하게 가져간다.(Stratified) + repeated
7     acc = cross_val_score(model,feature, y_train, scoring="accuracy", cv=rskfold).mean()
8     return acc
```

03 모델 선택

1. Logistic Regression
2. Naïve Bayes
3. Stochastic Gradient Descent
4. K-Nearest Neighbours
5. Decision Tree
6. Random Forest
7. Support Vector Machine
8. XGBoost
9. LightGBM
10. Extratree

GridSearchCV로 하이퍼 파라미터 튜닝 후 성능 비교

* GridSearchCV
: 분류, 회귀 알고리즘에 사용되는 하이퍼파라미터를 순차적으로 입력해 학습하면 자동으로 가장 좋은 파라미터 알려줌

03

모델 선택

1. Logistic Regression

```
from sklearn.linear_model import LogisticRegression

# 모델 생성 및 학습 시키기
logistic = LogisticRegression(random_state=42)
acc_cv = scoring(logistic,X_train_standard)
print("acc(Cross-validation): ",acc_cv)

new_row = {"Model":"Logisitic Regression","Accuracy(Cross-validation)": acc_cv,"Note":"x_standard-scaling"}
models= models.append(new_row,ignore_index = True)
```

Accuracy: 0.973184

2. Naïve Bayes

```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
acc_cv = scoring(gnb,X_train_standard)
print("acc(Cross-validation): ",acc_cv)


new_row = {"Model":"GaussianNB","Accuracy(Cross-validation)": acc_cv,"Note":"x_standard-scaling"}
models= models.append(new_row,ignore_index = True)
```

Accuracy: 0.869928

3. Stochastic Gradient Descent

```
from sklearn.linear_model import SGDClassifier

sgd = SGDClassifier(random_state = 42)

parameters={'loss': ['hinge', 'modified_huber', 'log'],
            "alpha": np.logspace(-4,4,1),
            "penalty": ["l1","l2", 'elasticnet']}

rskfold = RepeatedStratifiedKFold(n_splits=5, n_repeats=5, random_state=42)
grid_search = GridSearchCV(sgd,parameters, cv=rskfold, scoring='accuracy')

grid_search.fit(X_train_minmax,y_train)


print('최적의 하이퍼파라미터',grid_search.best_params_)
print('최적 모델의 cv score', grid_search.best_score_)
print('최적 모델',grid_search.best_estimator_ )
```

Accuracy: 0.974092

4. K-Nearest Neighbours

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
acc_cv = scoring(knn,X_train_robust)
print("acc(Cross-validation): ",acc_cv)


new_row = {"Model":"K-Nearest_Neighbours","Accuracy(Cross-validation)": acc_cv,"Note":"x_robust-scaling"}
models= models.append(new_row,ignore_index = True)
```

Accuracy: 0.909687

03 모델 선택

5. Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(random_state=42)

parameters = {"min_samples_leaf":[1, 2, 3, 4, 5, 6, 7, 8, 9],
              "max_depth":[2, 3, 4, 5, 6, None],
              "min_samples_split":[2, 3, 4, 5, 6, 7, 8, 9, 10]}

rskfold = RepeatedStratifiedKFold(n_splits=5, n_repeats=5, random_state=42)
grid_search = GridSearchCV(dt,parameters, cv=rskfold, scoring='accuracy')

grid_search.fit(X_train_minmax,y_train)


print('최적의 하이퍼파라미터',grid_search.best_params_)
print('최적 모델의 cv score', grid_search.best_score_)
print('최적 모델',grid_search.best_estimator_ )
```

Accuracy: 0.942138

6. RandomForest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

rf_class= RandomForestClassifier(random_state=42,n_estimators=500)

parameters ={
             'max_depth': [9,10,-1],#트리의 최대 깊이
             'min_samples_leaf': [3], #하나의 리프에 있어야 하는 최소 샘플 수
             'min_samples_split': [2]} #split을 하기 위한 최소 sample 수

rskfold = RepeatedStratifiedKFold(n_splits=5, n_repeats=5, random_state=42)
grid_search = GridSearchCV(rf_class,parameters, cv=rskfold, scoring='accuracy')

grid_search.fit(X_train,y_train)
```

Accuracy: 0.996300

7. Support Vector Machine

```
from sklearn import svm
svc = svm.SVC(kernel="rbf",random_state=42) #Support Vector Classification

acc_cv = scoring(svc,X_train_standard)

print("acc(Cross-validation): ",acc_cv)

new_row = {"Model":"SVM","Accuracy(Cross-validation)": acc_cv,"Note":"x_standard-scaling"}
models= models.append(new_row,ignore_index = True)
#행 추가할때 반드시 ignore_index = True 적어야 함.
```

Accuracy: 0.982427

8. XGBboost

```
from xgboost import XGBClassifier

xgb=XGBClassifier(n_estimators=300,random_state=42)

parameters = {
        'min_child_weight': [1, 5],
        'gamma': [0.5, 1, 1.5, 2],
        'subsample': [0.6, 0.8],
        'colsample_bytree': [0.6, 0.8],
        'max_depth': [3, 4]
        }

rskfold = RepeatedStratifiedKFold(n_splits=5, n_repeats=5, random_state=42)
grid_search = GridSearchCV(xgb,parameters, cv=rskfold, scoring='accuracy')

grid_search.fit(X_train,y_train)
```

Accuracy: 0.986127

03 모델 선택

9. LightGBM

```
from sklearn.model_selection import GridSearchCV

lgbm = LGBMClassifier(random_state=42)

parameters ={'max_depth': [5,10,30],
             'learning_rate': [0.01,0.05,0.1],
             'num_iterations':[500,1000,2000]}

rskfold = RepeatedStratifiedKFold(n_splits=5, n_repeats=5, random_state=42)
grid_search = GridSearchCV(lgbm, parameters,cv=rskfold, scoring="accuracy")

grid_search.fit(X_train,y_train)

print('최적의 하이퍼파라미터',grid_search.best_params_)
print('최적 모델의 cv score', grid_search.best_score_)
print('최적 모델',grid_search.best_estimator_ )
```

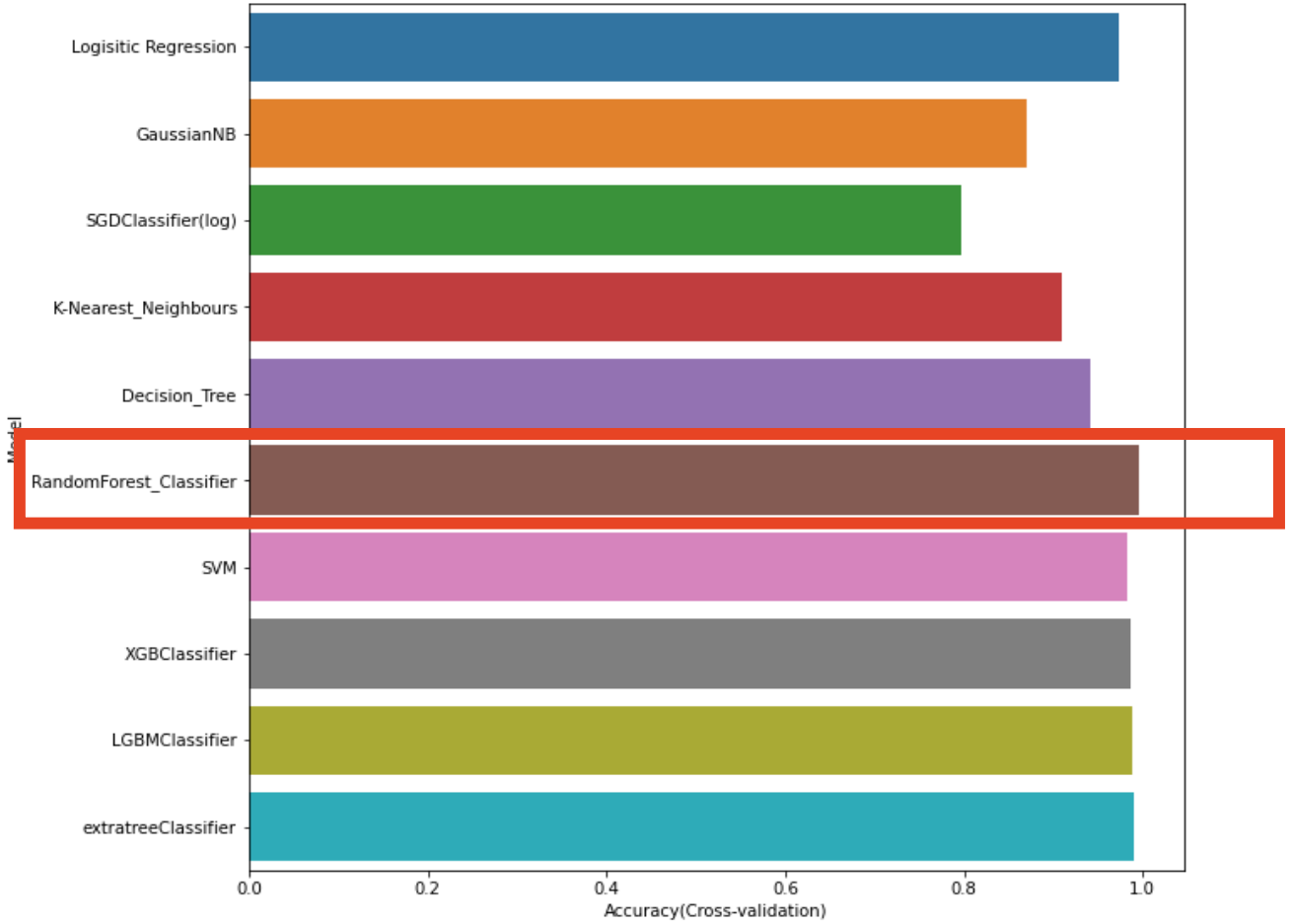
Accuracy: 0.993986

10. ExtraTREE

```
from sklearn.ensemble import ExtraTreesClassifier
xtree = ExtraTreesClassifier(n_estimators=2000, random_state=42, n_jobs =-1)

parameters ={'max_depth': [20,30,40]}  #리프의 총 수}


rskfold = RepeatedStratifiedKFold(n_splits=5, n_repeats=5, random_state=42)
grid_search = GridSearchCV(xtree, parameters,cv=rskfold, scoring="accuracy")

grid_search.fit(X_train_minmax,y_train)

print('최적의 하이퍼파라미터',grid_search.best_params_)
print('최적 모델의 cv score', grid_search.best_score_)
print('최적 모델',grid_search.best_estimator_ )

```

Accuracy: 0.990756

03 모델 선택

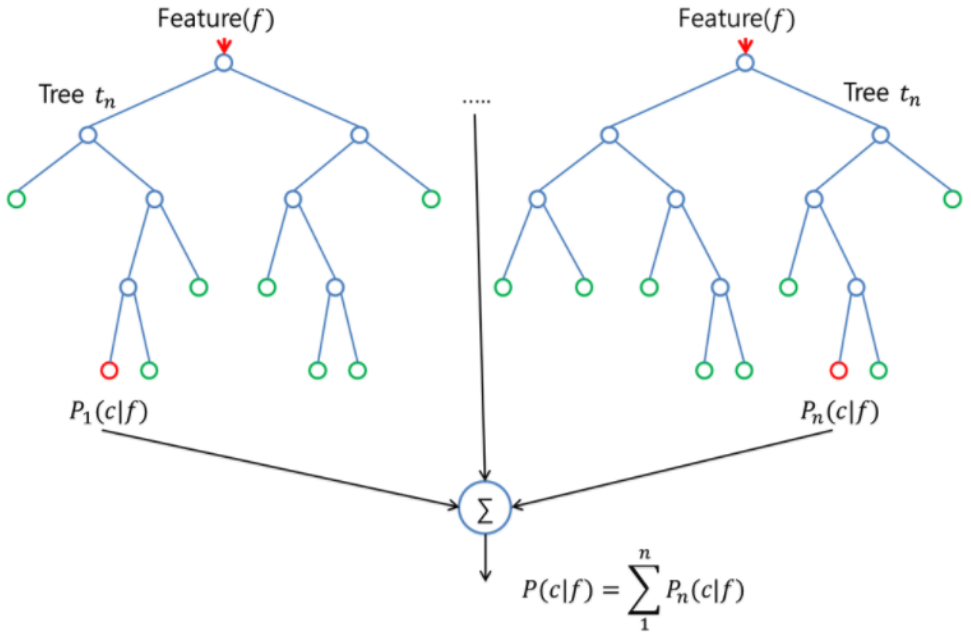


Accuracy 기준 가장 높은 성능을 보여준
RandomForest를
최종 모델로 선정!

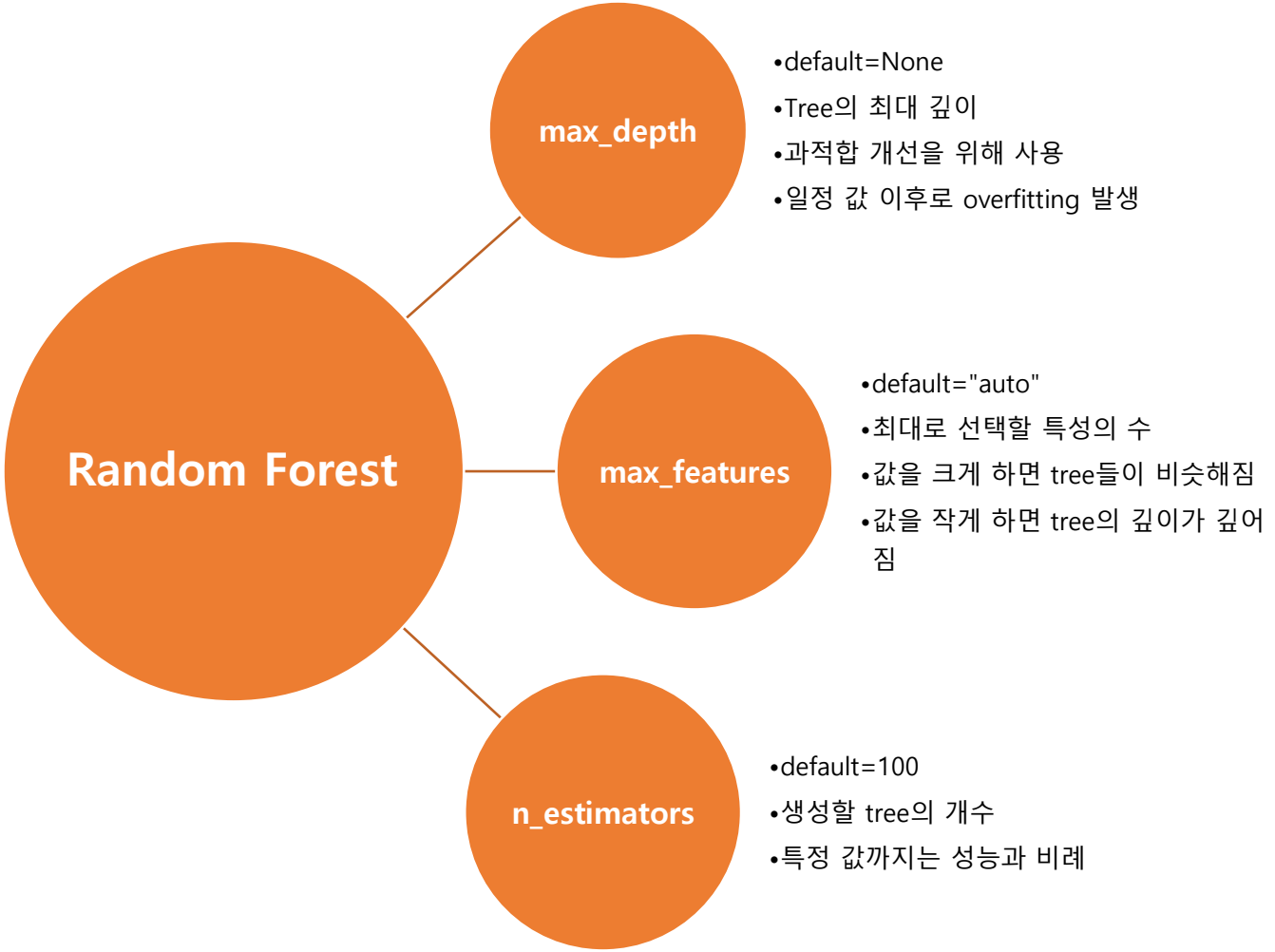
Accuracy: 0.996300

Random Forest

- Decision Tree를 기반으로 한 앙상블 머신러닝 모델
- 앙상블 방법 중 배깅(훈련 세트에서 중복 허용하여 샘플링 하는 방식)을 사용
- 여러 개의 tree를 형성하고 데이터를 각 트리에 동시에 통과시킨 후, 가장 많은 득표결과를 최종결과로 선택



Random Forest Parameters



04 Random Forest Parameters 세부 튜닝

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

rf_class= RandomForestClassifier(random_state=42)

parameters ={'n_estimators': [80, 90, 100, 110, 120],
             'max_features':[1, 2, 3, 4, 5, 'auto'],
             'max_depth':[2, 3, 4, 5, 6, None]}


rskfold = RepeatedStratifiedKFold(n_splits=5, n_repeats=5, random_state=42)
grid_search = GridSearchCV(rf_class,parameters, cv=rskfold, scoring='accuracy')

grid_search.fit(X_train,y_train)


print('최적의 하이퍼파라미터',grid_search.best_params_)
print('최적 모델의 cv score', grid_search.best_score_)
print('최적 모델',grid_search.best_estimator_ )

```

04 Random Forest Parameters 세부 튜닝

```python
from sklearn.ensemble import RandomForestClassifier
rf_class= RandomForestClassifier(random_state=42, max_depth = None,
                                 max_features = 5, n_estimators = 100)

acc_cv = scoring(rf_class,X_train)
print("acc(Cross-validation): ",acc_cv)

rf_class.fit(X_train,y_train)
rf_train_acc = 100 * rf_class.score(X_train, y_train)
rf_test_acc = 100 * rf_class.score(X_test, y_test)
print("train_acc: ",rf_train_acc)
print("test_acc: ",rf_test_acc)

new_row = {"Model":"RandomForest_Classifier","Accuracy(Cross-validation)": acc_cv,"Note":"base"}
models= models.append(new_row,ignore_index = True)
```

```
acc(Cross-validation):  0.9995402298850575
train_acc:  100.0
test_acc:  100.0
```

최종 모델의 Accuracy: 0.999540

Train_acc, test_acc 가 100프로이고 차이가 없으므로 overfitting은 일어나지 않았다!

04 AutoML

- Pycaret
  - ❑ 기존에 있던 Scikit-learn, XGBoost, LightGBM 등 여러가지 머신러닝 라이브러리를 총망라
  - ❑ Hight-Level API로 제공하여 단 몇 줄만에 데이터 분석, 머신러닝 모델 성능 비교 가능

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
knn	K Neighbors Classifier	0.9476	0.4513	1.0000	0.9476	0.9728	NaN	0.0000	0.1213
dummy	Dummy Classifier	0.9476	0.3667	1.0000	0.9476	0.9728	NaN	0.0000	0.0120
et	Extra Trees Classifier	0.9429	0.4051	0.9949	0.9473	0.9702	NaN	-0.0051	0.4787
svm	SVM - Linear Kernel	0.9381	0.0000	0.9799	0.9560	0.9673	NaN	0.1179	0.0153
rf	Random Forest Classifier	0.9381	0.3949	0.9901	0.9473	0.9677	NaN	-0.0051	0.4973
qda	Quadratic Discriminant Analysis	0.9381	0.3641	0.9901	0.9473	0.9677	NaN	-0.0051	0.0193
lightgbm	Light Gradient Boosting Machine	0.9381	0.4872	0.9850	0.9520	0.9675	NaN	0.0402	0.0880
catboost	CatBoost Classifier	0.9381	0.4923	0.9901	0.9473	0.9677	NaN	-0.0051	10.8460
ada	Ada Boost Classifier	0.9333	0.4564	0.9799	0.9512	0.9646	NaN	0.0540	0.1300
xgboost	Extreme Gradient Boosting	0.9333	0.4974	0.9799	0.9516	0.9646	NaN	0.0377	4.3307
gbc	Gradient Boosting Classifier	0.9095	0.4128	0.9597	0.9453	0.9513	NaN	-0.0254	0.2453
ridge	Ridge Classifier	0.9000	0.0000	0.9495	0.9443	0.9453	NaN	-0.0296	0.0153
dt	Decision Tree Classifier	0.8905	0.3769	0.9348	0.9489	0.9396	NaN	0.0286	0.0180
lr	Logistic Regression	0.8810	0.4026	0.9198	0.9533	0.9352	NaN	0.0688	0.2880
lda	Linear Discriminant Analysis	0.8762	0.5128	0.9136	0.9533	0.9308	NaN	0.0269	0.0180
nb	Naive Bayes	0.6000	0.4949	0.5934	0.9755	0.7323	0.0776	0.1283	0.0147

<모델 성능 비교>

```
[ ]  1 top5_model
```

```
[KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                      metric_params=None, n_jobs=-1, n_neighbors=5, p=2,
                      weights='uniform'),
 ExtraTreesClassifier(bootstrap=False, ccp_alpha=0.0, class_weight=None,
                      criterion='gini', max_depth=None, max_features='auto',
                      max_leaf_nodes=None, max_samples=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=-1,
                      oob_score=False, random_state=1776, verbose=0,
                      warm_start=False),
 SGDClassifier(alpha=0.0001, average=False, class_weight=None,
               early_stopping=False, epsilon=0.1, eta0=0.001, fit_intercept=True,
               l1_ratio=0.15, learning_rate='optimal', loss='hinge',
               max_iter=1000, n_iter_no_change=5, n_jobs=-1, penalty='l2',
               power_t=0.5, random_state=1776, shuffle=True, tol=0.001,
               validation_fraction=0.1, verbose=0, warm_start=False),
 RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                        criterion='gini', max_depth=None, max_features='auto',
                        max_leaf_nodes=None, max_samples=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=100,
                        n_jobs=-1, oob_score=False, random_state=1776, verbose=0,
                        warm_start=False)]
```

<TOP 5 Model>

04 AutoML

```
tuned_top5 = [tune_model(i,n_iter = 10) for i in top5_model]   # top11 모델에 대해 각각 튜닝을 진행
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	1.0000	0.0000	1.0	1.0000	1.0000	NaN	0.0
1	1.0000	0.0000	1.0	1.0000	1.0000	NaN	0.0
2	1.0000	0.0000	1.0	1.0000	1.0000	NaN	0.0
3	1.0000	0.0000	1.0	1.0000	1.0000	NaN	0.0
4	0.9286	0.7692	1.0	0.9286	0.9630	0.0	0.0
5	0.9286	1.0000	1.0	0.9286	0.9630	0.0	0.0
6	0.9286	0.7692	1.0	0.9286	0.9630	0.0	0.0
7	0.9286	1.0000	1.0	0.9286	0.9630	0.0	0.0
8	0.9286	0.2308	1.0	0.9286	0.9630	0.0	0.0
9	0.9286	1.0000	1.0	0.9286	0.9630	0.0	0.0
10	0.9286	0.0385	1.0	0.9286	0.9630	0.0	0.0
11	0.9286	0.0769	1.0	0.9286	0.9630	0.0	0.0
12	0.9286	1.0000	1.0	0.9286	0.9630	0.0	0.0
13	0.9286	0.0769	1.0	0.9286	0.9630	0.0	0.0
14	0.9286	0.3077	1.0	0.9286	0.9630	0.0	0.0
Mean	0.9476	0.4179	1.0	0.9476	0.9728	NaN	0.0
SD	0.0316	0.4262	0.0	0.0316	0.0164	NaN	0.0

<TOP 모델 튜닝 진행>

하이퍼파라미터 튜닝 이외에
성능을 좋게 만드는 방법은 없을까?

Voting, Stacking 사용해보자!

Voting: weak learner의 예측값 다수결 투표 혹은 가중치 합 이용
Stacking: weak learner들의 예측결과로 재학습해 최종 예측값 결정

04 AutoML

```
blend5_hard = blend_models(estimator_list=tuned_top5, method='hard', choose_better = True)   # 위의 best 3 모델에 대해 soft voting 방법으로 최적화
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	1.0000	0.0	1.0	1.0000	1.0000	NaN	0.0
1	1.0000	0.0	1.0	1.0000	1.0000	NaN	0.0
2	1.0000	0.0	1.0	1.0000	1.0000	NaN	0.0
3	1.0000	0.0	1.0	1.0000	1.0000	NaN	0.0
4	0.9286	0.0	1.0	0.9286	0.9630	0.0	0.0
5	0.9286	0.0	1.0	0.9286	0.9630	0.0	0.0
6	0.9286	0.0	1.0	0.9286	0.9630	0.0	0.0
7	0.9286	0.0	1.0	0.9286	0.9630	0.0	0.0
8	0.9286	0.0	1.0	0.9286	0.9630	0.0	0.0
9	0.9286	0.0	1.0	0.9286	0.9630	0.0	0.0
10	0.9286	0.0	1.0	0.9286	0.9630	0.0	0.0
11	0.9286	0.0	1.0	0.9286	0.9630	0.0	0.0
12	0.9286	0.0	1.0	0.9286	0.9630	0.0	0.0
13	0.9286	0.0	1.0	0.9286	0.9630	0.0	0.0
14	0.9286	0.0	1.0	0.9286	0.9630	0.0	0.0
Mean	0.9476	0.0	1.0	0.9476	0.9728	NaN	0.0
SD	0.0316	0.0	0.0	0.0316	0.0164	NaN	0.0

<Tuning된 Top Model Soft Voting>

```
blend5_stack = stack_models(estimator_list=tuned_top5)
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.0000	0.0000	0.0	0.0	0.0	0.0	0.0
1	0.0000	0.0000	0.0	0.0	0.0	0.0	0.0
2	0.0000	0.0000	0.0	0.0	0.0	0.0	0.0
3	0.0000	0.0000	0.0	0.0	0.0	0.0	0.0
4	0.0714	0.5000	0.0	0.0	0.0	0.0	0.0
5	0.0714	0.5000	0.0	0.0	0.0	0.0	0.0
6	0.0714	0.5000	0.0	0.0	0.0	0.0	0.0
7	0.0714	0.5000	0.0	0.0	0.0	0.0	0.0
8	0.0714	0.5000	0.0	0.0	0.0	0.0	0.0
9	0.0714	0.5000	0.0	0.0	0.0	0.0	0.0
10	0.0714	0.5000	0.0	0.0	0.0	0.0	0.0
11	0.0714	0.5000	0.0	0.0	0.0	0.0	0.0
12	0.0714	0.5000	0.0	0.0	0.0	0.0	0.0
13	0.0714	0.5000	0.0	0.0	0.0	0.0	0.0
14	0.0714	0.5000	0.0	0.0	0.0	0.0	0.0
Mean	0.0524	0.3667	0.0	0.0	0.0	0.0	0.0
SD	0.0316	0.2211	0.0	0.0	0.0	0.0	0.0

<Top Model Stacking>

04

최종 결과

159	춘천구조대장	춘천	0.87301	3	하루 전

05

한계 및 의의

01

시간/제출 횟수 제약

여유가 있었더라면
더 다양한 Scaling기법,
tuning을
활용해보고 싶음

02

과적합 문제

Oversampling으로 인한
overfitting 문제를 완벽하
게 해결하지 못한 아쉬움

03

결측치 처리

Random forest imputer
이외에도
다양한 imputer를 시도 해
보고 싶다

04

전처리 차이

같은 feature라도
train/test에 따라 차이가
있을 수 있다

05

AUTO ML

결과적으로는 성능을 더 높여
주지 못했지만, 해로운 경험.
성능을 높여볼 수 있는 또 다
른 방법을 알게 됨

경청해주셔서 감사합니다 :D