IIE3104 과제2 분석보고서

2021190002 장서현

I. 데이터 설명

반도체 공정에서 웨이퍼의 품질 상태를 판정하기 위한 데이터이다. 총 474개의 독립 변수와 웨이퍼의 품질 상태를 나타내는 (정상: -1, 불량:1) 1개의 종속변수로 구성되어 있다. 전체 데이터 개수는 254개로 전형적인 'Large p Small n' 문제로 학습 중 과적합을 방지하는 것이 중요할 것으로 예상된다.

```
train_X = train_set.drop('Y', axis = 1)
train_Y = train_set['Y']

train_X.isna().sum() # 행별 결측치 개수 관측
```

✓ 0.0s

```
X1      0
X2      0
X3      0
X4      0
X5      0
       ..
X470    0
X471    0
X472    0
X473    0
X474    0
Length: 474, dtype: int64
```

행별 데이터 관측을 통해 데이터셋에 결측치가 없음을 확인하였다.

II. 전처리 과정

[데이터 특성 파악]

```
pd.DataFrame({
    'Min': train_X.min(axis=0),
    'Max': train_X.max(axis=0),
    'Mean': train_X.mean(axis=0),
    'Std': train_X.std(axis=0)
})
```

✓ 0.0s

	Min	Max	Mean	Std
X1	0.036582	1.000000	0.395527	0.166594
X2	0.056331	1.000000	0.419683	0.151138
X3	0.094992	1.000000	0.372956	0.166990
X4	0.000000	1.000000	0.305515	0.226031
X5	0.000000	1.000000	0.176096	0.092270
...	...	...	...	...
X470	0.000000	1.000000	0.019682	0.070269
X471	0.000000	0.936893	0.391774	0.163002
X472	0.000000	1.000000	0.272208	0.166417
X473	0.000000	1.000000	0.311662	0.193166
X474	0.000000	1.000000	0.241088	0.185899

474 rows × 4 columns

데이터의 대표적 특성인 최대값, 최소값, 평균, 표준편차를 확인해보니, 대부분의 데이터가 0과 1 사이의 값을 가지며, 평균과 표준편차가 유사한 변수들이 많다는 것을 인지하였다.

[이상치 제거]

대부분의 변수들은 평균을 대략 0.3이지만, 소수의 변수들은 평균이 0.05 이하이기도 하였다. 샘플 값이 대부분 0이며 극소수의 샘플이 1인 경우 평균이 0.05 이하로 발생하였는데, 결국은 나머지 변수들과 같이 [0,1] 사이의 값을 가졌기 때문에 이상치로 판단하기 어려웠다. 주어진 샘플 수가 적은 것 또한 샘플을 제거하지 않은 이유 중 하나이다. 결론적으로, 이상치 제거는 하지 않기로 결정하였다.

[스케일링 기법]

```python
# Min-Max scaler version
X_train, X_valid, Y_train, Y_valid = train_test_split(train_X, train_Y, test_size = 0.2, shuffle = True, stratify = train_Y, random_state = RANDOM_SEED)
scaler = MinMaxScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = train_X.columns)
X_valid = pd.DataFrame(scaler.transform(X_valid), columns = train_X.columns)
test_X = pd.DataFrame(scaler.transform(test_X), columns = train_X.columns)
```
✓ 0.0s

대부분의 데이터가 최솟값을 0으로, 최대값을 1로 대략적으로 정형화가 되어있었다. 그렇기에 스케일링을 아예 하지 않는 방안도 고려했지만, 모든 변수들에 대한 통일성이 있는 계산을 위해 Min-Max Scaler를 활용하여 데이터 값들을 [0,1] 사이에 위치하도록 데이터 스케일링을 진행하였다. 실제 성능 검증 결과, Min-Max 스케일링을 진행하였을 때 가장 좋은 성능이 나왔다.

III. 변수 선택, 선정 이유

```python
# int64 타입의 컬럼 출력
int_columns = train_X.select_dtypes(include=['int64']).columns
print("int64 타입 컬럼:")
print(int_columns)

# int64 타입 컬럼들의 값이 모두 0인지 확인
print("\ntrain 데이터의 int64 컬럼 값 확인:")
for col in int_columns:
    print(f"{col}의 unique 값:", train_X[col].unique())

# int64 타입의 컬럼 제거
train_X = train_X.select_dtypes(exclude=['int64'])
test_X = test_X.select_dtypes(exclude=['int64'])

print("\nint64 타입 컬럼 제거 후 데이터 형태:")
print("train_X 형태:", train_X.shape)
print("test_X 형태:", test_X.shape)
```
✓ 0.0s

```
int64 타입 컬럼:
Index(['X69', 'X189', 'X192', 'X288', 'X293', 'X389'], dtype='object')

train 데이터의 int64 컬럼 값 확인:
X69의 unique 값: [0]
X189의 unique 값: [0]
X192의 unique 값: [0]
X288의 unique 값: [0]
X293의 unique 값: [0]
X389의 unique 값: [0]

int64 타입 컬럼 제거 후 데이터 형태:
train_X 형태: (203, 468)
```

데이터 타입을 확인한 결과, 대부분은 float형 데이터였다. 6개의 변수들의 데이터 값만이 int64형을 띄었고, 모두 값이 0임을 확인하여, 해당 6개의 변수들을 삭제하였다.

```python
# Grid Search를 위한 파라미터 설정
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100]  # C는 규제 강도의 역수
}

# Grid Search 수행
grid_search = GridSearchCV(
    lasso,
    param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)
grid_search.fit(X_train, Y_train)

# 최적의 파라미터로 모델 생성
best_lasso = LogisticRegression(
    penalty='l1',
    solver='liblinear',
    C=grid_search.best_params_['C'],
    random_state=RANDOM_SEED
)
best_lasso.fit(X_train, Y_train)
```

```python
# 변수 중요도 기반 특성 선택
selector = SelectFromModel(best_lasso, prefit=True)

# 선택된 특성의 인덱스 추출
selected_features_lasso = X_train.columns[selector.get_support()]

print("\nLasso로 선택된 변수 개수:", len(selected_features_lasso))
print("\nLasso로 선택된 변수:")
print(selected_features_lasso)

# 선택된 변수만 사용하여 데이터셋 변환
X_train_lasso = X_train[selected_features_lasso]
X_valid_lasso = X_valid[selected_features_lasso]
test_X_lasso = test_X[selected_features_lasso]
```

다음으로, mRMR, SVM-RFE, PCA, Ridge, Elastic-Net과 같은 차원 축소 방법을 모두 고려하였지만 (Appendix 참고), 나머지 468개의 변수들의 평균과 표준편차가 유사한 변수들이 많다는 사실과 변수 개수 자체가 상당히 크다는 사실에 기반하여, LASSO를 사용하기로 결정하였다. LASSO로 차원축소를 함으로써 유사한 변수들 중에 Y와의 상관성이 높은 중요한 변수들을 선택하고, 학습데이터에게 과적합되는 현상이 방지되기를 기대하였다. 또, Grid Search를 통해 최적의 파라미터를 결정하였다. 그 결과, 총 35개의 변수가 선택되었고, 이를 최종 모델의 변수들로 사용하였다.

```
    print("최종적으로 선택된 변수들:")
    print(list(X_train_lasso.columns))
    print(f"선택된 변수의 개수: {len(list(X_train_lasso.columns))}")
```
✓ 0.0s

```
최종적으로 선택된 변수들:
['X8', 'X16', 'X22', 'X27', 'X36', 'X41', 'X48', 'X70', 'X86', 'X97',
선택된 변수의 개수: 35
```

IV. 교차 검증

```
rf = RandomForestClassifier()
rnd_search = RandomizedSearchCV(
    estimator = rf,
    param_distributions = rf_params,
    scoring = 'f1_weighted',
    n_iter = 10,
    cv = StratifiedKFold(n_splits = 5),
    random_state = RANDOM_SEED,
)
```

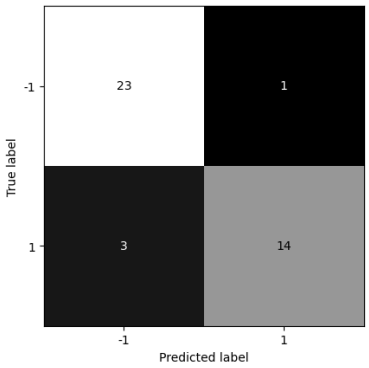
모든 모델에서 Stratified K-Fold (n_splits = 5)를 사용하여 교차 검증을 진행하였다. 하나의 fold 안에 있는 클래스 비율이 전체 클래스 비율과 동일하게 설정하여, 데이터가 편향되지 않게 끔 하였다.

V. 하이퍼파라미터 최적화 방법

[Random Forest]

```
rf_params = {
    'n_estimators': [100, 300, 500, 700, 900],
    'max_depth': [5, 10, 20, 30, 50],
    'min_samples_split': [2, 4, 6, 8, 10],
    'min_samples_leaf': [0.3, 0.5, 1, 2, 4],
}

rf = RandomForestClassifier()
rnd_search = RandomizedSearchCV(
    estimator = rf,
    param_distributions = rf_params,
    scoring = 'f1_weighted',
    n_iter = 10,
    cv = StratifiedKFold(n_splits = 5),
    random_state = RANDOM_SEED,
)

cv_res = rnd_search.fit(X_train, Y_train)
print("Hyperparameter:", cv_res.best_params_)
print("F1-Weighted:", cv_res.best_score_)
```
✓ 19.6s

```
Hyperparameter: {'n_estimators': 700, 'min_samples_split': 6, 'min_samples_leaf': 1, 'max_depth': 20}
F1-Weighted: 0.8597375573428788
```

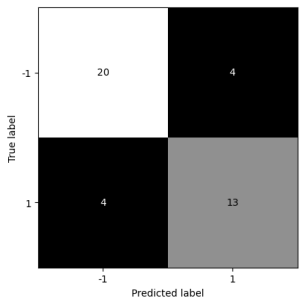


Random Forest 모델의 성능을 최적화하기 위해 RandomizedSearchCV를 활용하여, n_estimators(트리 개수), max_depth(최대 깊이), min_samples_split(노드 분할 최소 샘플 수), min_samples_leaf(리프 노드 최소 샘플 수) 등의 하이퍼 파라미터를 튜닝하였다. RandomizedSearchCV는 설정된 하이퍼파라미터 공간에서 10번의 샘플링을 수행하여 n_estimators=700, max_depth=20, min_samples_split=6, min_samples_leaf=1의 최적 조합을 도출하였으며, 최적화된 모델의 F1-Score는 0.8597로 나타났다.

[Gradient Boosting]

```
import numpy as np
gbm_params = {
    'learning_rate': np.arange(0.01, 0.5, 0.01),
    'n_estimators': np.arange(100, 1000, 100),
    'max_depth': [1, 3, 7, 10],
    'min_samples_split': np.arange(0.2, 1.2, 0.2),
    'min_samples_leaf': np.arange(0.1, 1.0, 0.2),
    'subsample': np.arange(0.2, 1.2, 0.2),
}

gbm = GradientBoostingClassifier()
rnd_search = RandomizedSearchCV(
    estimator=gbm,
    param_distributions=gbm_params,
    scoring='f1_weighted',
    n_iter=10,
    cv=StratifiedKFold(n_splits=5),
    random_state=RANDOM_SEED,
)
cv_res = rnd_search.fit(X_train, Y_train)
print("Hyperparameter:", cv_res.best_params_)
print("F1-Weighted:", cv_res.best_score_)
```
✓ 10.3s

```
Hyperparameter: {'subsample': 0.6000000000000001, 'n_estimators': 800, 'min_samples_split': 0.4, 'min_samples_leaf': 0.1, 'max_depth': 3, 'learning_rate': 0.37}
F1-Weighted: 0.8508012256420218
```

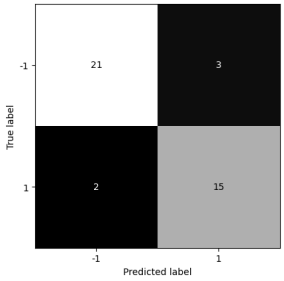


Gradient Boosting 모델의 성능을 최적화하기 위해 RandomizedSearchCV를 활용하여,

learning_rate(학습률), n_estimators(트리 개수), max_depth(최대 깊이), min_samples_split(노드 분할 최소 샘플 수), min_samples_leaf(리프 노드 최소 샘플 수), subsample(샘플링 비율) 등의 하이퍼파라미터를 튜닝하였다. 10번의 샘플링을 수행 결과, learning_rate=0.37, n_estimators=800, max_depth=3, min_samples_split=0.4, min_samples_leaf=0.1, subsample=0.6의 최적 조합을 도출하였으며, 최적화된 모델의 F1-Score는 0.8508로 나타났다.

[Light GBM]

```
lgb_params ={
    'learning_rate': [0.015, 0.025, 0.05, 0.1, 0.2],
    'num_iterations': [100, 300, 500, 700 ,900],
    'num_leaves': [2, 5, 10, 20, 50],
    'min_child_samples': [20, 50, 100, 500, 1000],
    'max_depth': [1, 3, 5, 7, 9],
}

lgbm = lgb.LGBMClassifier(verbose = 0)
rnd_search = RandomizedSearchCV(
    estimator = lgbm,
    param_distributions = lgb_params,
    scoring = 'f1_weighted',
    n_iter = 10,
    cv=StratifiedKFold(n_splits = 5),
    random_state = RANDOM_SEED,
)

cv_res = rnd_search.fit(X_train, Y_train)
print("Hyperparameter:", cv_res.best_params_)
print("F1-Weighted:", cv_res.best_score_)
```

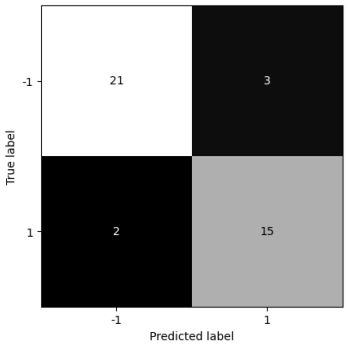



```
Hyperparameter: {'num_leaves': 20, 'num_iterations': 300, 'min_child_samples': 20, 'max_depth': 3, 'learning_rate': 0.1}
F1-Weighted: 0.8477630633802695
```

LightGBM 모델의 성능을 최적화하기 위해 RandomizedSearchCV를 활용하여, learning_rate(학습률), num_iterations(부스팅 반복 횟수), num_leaves(리프의 최대 개수), min_child_samples(리프 노드의 최소 샘플 수), max_depth(트리의 최대 깊이) 등의 하이퍼파라미터를 튜닝하였다. RandomizedSearchCV는 설정된 하이퍼파라미터 공간에서 10번의 샘플링을 수행하여 learning_rate=0.1, num_iterations=300, num_leaves=20, min_child_samples=20, max_depth=3의 최적 조합을 도출하였으며, 최적화된 모델의 F1-Score는 0.8478로 나타났다.

[XGBoost]

```
xgb_params = {
    'learning_rate': [0.015, 0.025, 0.05, 0.1, 0.2],
    'gamma': [0.05, 0.1, 0.2, 0.5, 1.0],
    'max_depth': [2, 5, 10, 15, 20],
    'lambda': [0.05, 0.1, 0.2, 0.5, 1.0],
    'alpha': [0.05, 0.1, 0.2, 0.5, 1.0],
}

#[-1,1] to [0,1]
Y_train_xgb = (Y_train + 1) // 2

xgbm = xgb.XGBClassifier(verbose = 0)
rnd_search = RandomizedSearchCV(
    estimator = xgbm,
    param_distributions = xgb_params,
    scoring = 'f1_weighted',
    n_iter = 10,
    cv=StratifiedKFold(n_splits = 5),
    random_state = RANDOM_SEED,
)

cv_res = rnd_search.fit(X_train, Y_train_xgb)
print("Hyperparameter:", cv_res.best_params_)
print("F1-Weighted:", cv_res.best_score_)
✓ 2.8s
Hyperparameter: {'max_depth': 20, 'learning_rate': 0.05, 'lambda': 0.2, 'gamma': 0.2, 'alpha': 0.05}
F1-Weighted: 0.8049568029154223
```



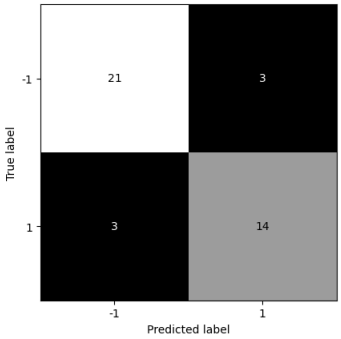
XGBoost 분류기의 성능을 최적화하기 위해 RandomizedSearchCV를 활용하여, learning_rate(학습률), max_depth(최대 깊이), lambda(정규화 항), gamma(노드 분할 최소 손실 감소), alpha(정규화 항) 등의 하이퍼파라미터를 튜닝하였다. 10번의 샘플링 후, learning_rate=0.05, max_depth=20, lambda=0.2, gamma=0.2, alpha=0.05의 최적 조합을 도출하였으며, 최적화된 모델의 F1-Score는 0.805로 나타났다.

[CatBoost]

```python
cat_params = {
    'learning_rate': [0.015, 0.025, 0.05, 0.1, 0.2],
    'iterations': [100, 300, 500, 700, 900],
    'max_depth': [1, 2, 5, 10, 15],
    'min_data_in_leaf': [0.1, 0.3, 0.5, 0.7, 0.9],
}

cat = CatBoostClassifier(verbose = 0)
rnd_search = RandomizedSearchCV(
    estimator = cat,
    param_distributions = cat_params,
    scoring = 'f1_weighted',
    n_iter = 10,
    cv=StratifiedKFold(n_splits = 5),
    random_state = RANDOM_SEED,
)

cv_res = rnd_search.fit(X_train, Y_train)
print("Hyperparameter:", cv_res.best_params_)
print("F1-Weighted:", cv_res.best_score_)
```

✓ 2m 28.2s

```
Hyperparameter: {'min_data_in_leaf': 0.9, 'max_depth': 2, 'learning_rate': 0.05, 'iterations': 700}
F1-Weighted: 0.8475354191263282
```



CatBoost 분류기의 성능을 최적화하기 위해 RandomizedSearchCV를 활용하여, learning_rate(학습률), iterations(반복 횟수), max_depth(최대 깊이), min_data_in_leaf(리프 노드에 포함된 최소 데이터 수) 등의 하이퍼파라미터를 튜닝하였다. RandomizedSearchCV는 설정된 하이퍼파라미터 공간에서 10번의 샘플링을 수행하여 learning_rate=0.05, iterations=700, max_depth=2, min_data_in_leaf=0.9의 최적 조합을 도출하였으며, 최적화된 모델의 F1-Score는 0.8475로 나타났다.

VI. 결과

데이터의 전처리 없이, 제공된 474개의 독립변수를 모두 사용한 'original' 모델을 대조군으로 두었다.

Original					
Model	RF	GB	LightGBM	XGBoost	CatBoost
F1 score	0.8019	0.7593	0.7702	0.7686	0.8149

Original 모델에 비해, F1 score 값을 기준으로 바라보았을 때, 성능개선이 되었음을 확인할 수 있다.

Proposed					
Model	RF	GB	LightGBM	XGBoost	CatBoost
F1	0.8597	0.8568	0.8478	0.8050	0.8476

예측 결과는 아래와 같으며, 전체 예측값들은 '2021190002_장서현_prediction.csv' 파일에 첨부하였다.

```python
final = pd.DataFrame({'RF': rf_test, 'GBM': gb_test, 'LightGBM': lgb_test, 'XGBoost': xgb_test, 'CatBoost': cat_test})
final.to_csv('C:/Users/symply_jina/Desktop/2차 과제/2021190002_장서현_prediction.csv', index = False)
final
```

✓ 0.0s

	RF	GBM	LightGBM	XGBoost	CatBoost
0	1	1	1	1	1
1	1	1	1	1	1
2	-1	-1	-1	-1	-1
3	-1	-1	-1	-1	-1
4	1	1	1	1	1
5	1	1	1	1	1
6	-1	-1	1	1	-1
7	-1	-1	-1	1	-1
8	-1	-1	-1	-1	-1
9	1	1	1	1	1
10	1	1	1	1	1

VII. Appendix

[변수 선택 과정]

A. [SVM-RFE] 총 234개의 변수가 선택되었다.

```python
#SVM-RFE
from sklearn.svm import SVC
from sklearn.feature_selection import RFE

estimator = SVC(kernel = 'linear', random_state = RANDOM_SEED)
selector = RFE(estimator)
selector = selector.fit(X_train, Y_train)

X_train.columns[selector.support_]
```

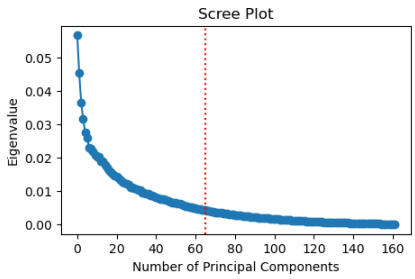
✓ 0.5s

```
Index(['X1', 'X2', 'X3', 'X4', 'X7', 'X8', 'X9', 'X11', 'X13', 'X14',
       ...
       'X452', 'X458', 'X460', 'X462', 'X464', 'X468', 'X469', 'X470', 'X471',
       'X472'],
      dtype='object', length=234)
```

```
Index(['X1', 'X2', 'X3', 'X4', 'X7', 'X8', 'X9', 'X11', 'X13', 'X14',
       ...
       'X452', 'X458', 'X460', 'X462', 'X464', 'X468', 'X469', 'X470', 'X471',
       'X472'],
      dtype='object', length=234)
```

B. [PCA] 65개의 잠재변수가 설명하는 분산이 대략 82%였다.

```python
# PCA - standard scaler
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_standard = scaler.fit_transform(X_train)
X_valid_standard = scaler.transform(X_valid)

n_features = X_train.shape[1]  # 변수 개수
n_samples = X_train.shape[0]   # 샘플 개수
n_components = min(n_features, n_samples)  # 최대 가능한 주성분 수

for i in range(n_components):
    pca = PCA(n_components = i + 1)
    pca.fit(X_train_standard)
    total_expl_var = sum(pca.explained_variance_ratio_) * 100
    print(i + 1, "개의 잠재변수가 설명하는 분산: %.2f%%" %total_expl_var)
```



C. [mRMR] 65개의 변수로 차원축소를 진행하였다.

```python
# mRMR

import numpy as np
k = 65
S = []

corr_matrix = (pd.concat([X_train, Y_train], axis = 1).corr().drop('Y'))
S.append(corr_matrix.index[np.argmax(corr_matrix['Y'])])

for i in range(1, k):
    relevance = np.abs(corr_matrix['Y'])
    redundancy = np.mean(np.abs(corr_matrix.loc[:, S]), axis = 1)
    candidate = (relevance - redundancy)
    candidate = candidate.drop(S)
    S.append(candidate.index[np.argmax(candidate)])
```

✓ 0.0s

[Original Version 코드]

A. 변수 선택 생략

```python
print("최종 선택 변수: ")
print(X_train.columns)
```

✓ 0.0s Python

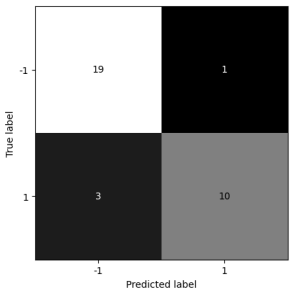
```
최종 선택 변수:
Index(['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10',
       ...
       'X465', 'X466', 'X467', 'X468', 'X469', 'X470', 'X471', 'X472', 'X473',
       'X474'],
      dtype='object', length=474)
```

B. Random Forest

```python
rf_params = {
    'n_estimators': [100, 300, 500, 700, 900],
    'max_depth': [5, 10, 20, 30, 50],
    'min_samples_split': [2, 4, 6, 8, 10],
    'min_samples_leaf': [0.3, 0.5, 1, 2, 4],
}

rf = RandomForestClassifier()
rnd_search = RandomizedSearchCV(
    estimator = rf,
    param_distributions = rf_params,
    scoring = 'f1_weighted',
    n_iter = 10,
    cv = StratifiedKFold(n_splits = 5),
    random_state = RANDOM_SEED,
)

cv_res = rnd_search.fit(X_train_original, Y_train)
print("Hyperparameter:", cv_res.best_params_)
print("F1-Weighted:", cv_res.best_score_)
```

✓ 28.1s

```
Hyperparameter: {'n_estimators': 500, 'min_samples_split': 8, 'min_samples_leaf': 1, 'max_depth': 20}
F1-Weighted: 0.8018644929544898
```

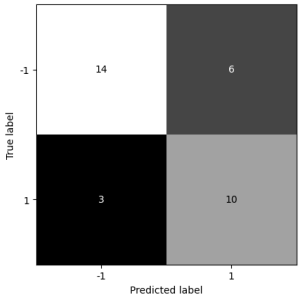


C. Gradient Boosting

```python
import numpy as np
gbm_params = {
    'learning_rate': np.arange(0.01, 0.5, 0.01),
    'n_estimators': np.arange(100, 1000, 100),
    'max_depth': [1, 3, 7, 10],
    'min_samples_split': np.arange(0.2, 1.2, 0.2),
    'min_samples_leaf': np.arange(0.1, 1.0, 0.2),
    'subsample': np.arange(0.2, 1.2, 0.2),
}

gbm = GradientBoostingClassifier()
rnd_search = RandomizedSearchCV(
    estimator=gbm,
    param_distributions=gbm_params,
    scoring='f1_weighted',
    n_iter=10,
    cv=StratifiedKFold(n_splits=5),
    random_state=RANDOM_SEED,
)
cv_res = rnd_search.fit(X_train_original, Y_train)
print("Hyperparameter:", cv_res.best_params_)
print("F1-Weighted:", cv_res.best_score_)
```

✓ 48.2s

```
Hyperparameter: {'subsample': 0.6000000000000001, 'n_estimators': 800, 'min_samples_split': 0.4, 'min_samples_leaf': 0.1, 'max_depth': 3,
F1-Weighted: 0.7593129754592709
```

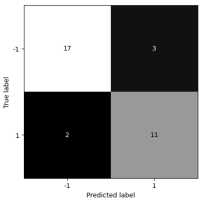


D. Light GBM

```
Hyperparameter: {'num_leaves': 10, 'num_iterations': 100, 'min_child_samples': 20, 'max_depth': 3, 'learning_rate': 0.05}
F1-Weighted: 0.7702400540635834
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
params = {'num_leaves': 10,
          'num_iterations': 100,
          'min_child_samples': 20,
          'max_depth': 3,
          'learning_rate': 0.05
          }
```

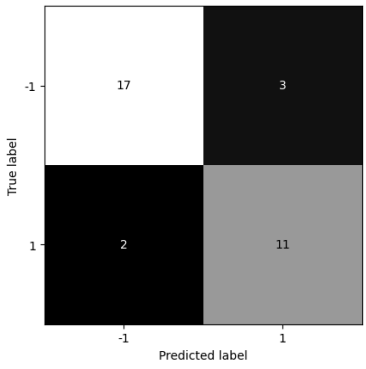


E. XG Boost

```python
xgb_params = {
    'learning_rate': [0.015, 0.025, 0.05, 0.1, 0.2],
    'gamma': [0.05, 0.1, 0.2, 0.5, 1.0],
    'max_depth': [2, 5, 10, 15, 20],
    'lambda': [0.05, 0.1, 0.2, 0.5, 1.0],
    'alpha': [0.05, 0.1, 0.2, 0.5, 1.0],
}

Y_train_xgb = (Y_train + 1) // 2

xgbm = xgb.XGBClassifier(verbose = 0)
rnd_search = RandomizedSearchCV(
    estimator = xgbm,
    param_distributions = xgb_params,
    scoring = 'f1_weighted',
    n_iter = 10,
    cv=StratifiedKFold(n_splits = 5),
    random_state = RANDOM_SEED,
)

cv_res = rnd_search.fit(X_train_original, Y_train_xgb)
print("Hyperparameter:", cv_res.best_params_)
print("F1-Weighted:", cv_res.best_score_)
```

✓ 8.6s

```
Hyperparameter: {'max_depth': 5, 'learning_rate': 0.05, 'lambda': 0.2, 'gamma': 0.2, 'alpha': 0.5}
F1-Weighted: 0.7686328251544721
```

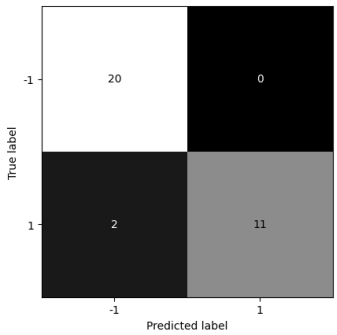


F. Cat Boost

```python
cat_params = {
    'learning_rate': [0.015, 0.025, 0.05, 0.1, 0.2],
    'iterations': [100, 300, 500, 700, 900],
    'max_depth': [1, 2, 5, 10, 15],
    'min_data_in_leaf': [0.1, 0.3, 0.5, 0.7, 0.9],
}

cat = CatBoostClassifier(verbose = 0)
rnd_search = RandomizedSearchCV(
    estimator = cat,
    param_distributions = cat_params,
    scoring = 'f1_weighted',
    n_iter = 10,
    cv=StratifiedKFold(n_splits = 5),
    random_state = RANDOM_SEED,
)

cv_res = rnd_search.fit(X_train_original, Y_train)
print("Hyperparameter:", cv_res.best_params_)
print("F1-Weighted:", cv_res.best_score_)
```

✓ 16m 2.1s

```
Hyperparameter: {'min_data_in_leaf': 0.7, 'max_depth': 5, 'learning_rate': 0.05, 'iterations': 500}
F1-Weighted: 0.8149167247662954
```



[성능 개선 과정] scaling은 1) no scaling, 2) min-max scaling, 3) standard scaling의 큰 틀 속에서 실험을 진행했으며, 변수 선택은 1) mRMR, 2) SVM-RFE, 3) Ridge, 4) Lasso, 5) Elastic-net를 활용하여 성능 개선을 시도하였다. 아래의 큰 틀에서의 업데이트 외에도 parameter tuning과 같은 미세한 튜닝도 진행한 뒤, 최종 모델을 선정하였다.

Update 1 (no-scaling + mRMR 85 + SVM-RFE = 총 259개 변수)					
Model	RF	GB	LightGBM	XGBoost	CatBoost
F1	0.8549	0.8222	0.8163	0.8190	0.8281

Update 2 (standard + mRMR 85 + SVM-RFE = 총 283개 변수)					
Model	RF	GB	LightGBM	XGBoost	CatBoost
F1	0.8583	0.7876	0.8271	0.8285	0.8373

Update 3 (min-max + mRMR 85 + SVM-RFE = 총 282개 변수)					
Model	RF	GB	LightGBM	XGBoost	CatBoost
F1	0.8574	0.8029	0.8403	0.8208	0.8251

Update 4 (min-max + ridge = 총 182개 변수)					
Model	RF	GB	LightGBM	XGBoost	CatBoost
F1	0.8645	0.8378	0.8123	0.8062	0.8278

Update 5 (min-max + lasso = 총 35개 변수) = Proposed Model					
Model	RF	GB	LightGBM	XGBoost	CatBoost
F1	0.8597	0.8568	0.8478	0.8050	0.8476

Update 6 (min-max + elastic = 총 121개 변수)					
Model	RF	GB	LightGBM	XGBoost	CatBoost
F1	0.8629	0.8236	0.8160	0.8205	0.8587

Update 7 (no scaling + ridge = 총 183개 변수)					
Model	RF	GB	LightGBM	XGBoost	CatBoost
F1	0.8378	0.8164	0.7992	0.8126	0.8295

Update 8 (no scaling + lasso = 총 40개 변수)					
Model	RF	GB	LightGBM	XGBoost	CatBoost
F1	0.8520	0.8320	0.8436	0.8373	0.8345

Update 9 (no scaling + elastic = 총 117개 변수)					
Model	RF	GB	LightGBM	XGBoost	CatBoost
F1	0.8648	0.8323	0.8428	0.8065	0.8590

Update 10 (standard + ridge = 총 176개 변수)					
Model	RF	GB	LightGBM	XGBoost	CatBoost
F1	0.8569	0.8427	0.8147	0.8255	0.8450

Update 11 (standard + lasso = 총 75개 변수)					
Model	RF	GB	LightGBM	XGBoost	CatBoost
F1	0.8707	0.8557	0.8227	0.8070	0.8508

Update 12 (standard + elastic = 총 121개 변수)					
Model	RF	GB	LightGBM	XGBoost	CatBoost
F1	0.8560	0.8549	0.8042	0.8205	0.8587