📚 [LAB09] 지도학습 > 분류 > 05-비선형 분류

📘 #01. 준비작업

📝 [1] 패키지 가져오기

```
# 라이브러리 기본 참조
from hossam import *
from pandas import DataFrame, concat
from matplotlib import pyplot as plt
import seaborn as sb
import numpy as np

from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV, learning_curve
from sklearn.preprocessing import StandardScaler

# 분류모형
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
```

```
📦 아이티윌 이광호 강사가 제작한 라이브러리를 사용중입니다.
📚 자세한 사용 방법은 https://py.hossam.kr 을 참고하세요.
📧 Email: leekh4232@gmail.com
🎬 Youtube: https://www.youtube.com/@hossam-codingclub
📝 Blog: https://blog.hossam.kr/
🩹 Version: 0.5.4
현재 설치된 'hossam' 패키지 버전: 0.5.4

✅ 시각화를 위한 한글 글꼴(NotoSansKR-Regular)이 자동 적용되었습니다.
```

📝 [2] 앞서 정의한 함수

함수 사용에 필요한 import문은 [1] 에서 생략되었습니다.

- hs_cls_bin_scores

📝 [3] 데이터 가져오기

```
origin = load_data("pima_indians_diabetes_preprocessed")
origin.head()
```

캐글에서 제공하는 pima_indians_diabetes의 전처리 완료 버전(결측치 정제+로그변환) (출처: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database)

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	1.946	5.004	72	3.584	4.836	3.544	0.487	3.932	1
1	0.693	4.454	66	3.401	4.836	3.318	0.301	3.466	0
2	2.197	5.215	64	3.401	4.836	3.190	0.514	3.497	1
3	0.693	4.500	66	3.178	4.554	3.371	0.154	3.091	0
4	0.000	4.927	40	3.584	5.130	3.786	1.190	3.526	1

📝 [4] 훈련,검증 데이터 분리

```
df = origin.copy()

# 중요!!! 종속변수를 정수형으로 변환해야 한다.
df['Outcome'] = df['Outcome'].astype('int')

yname = "Outcome"
x = df.drop(columns=[yname])
y = df[yname]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=52)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((614, 8), (154, 8), (614,), (154,))
```

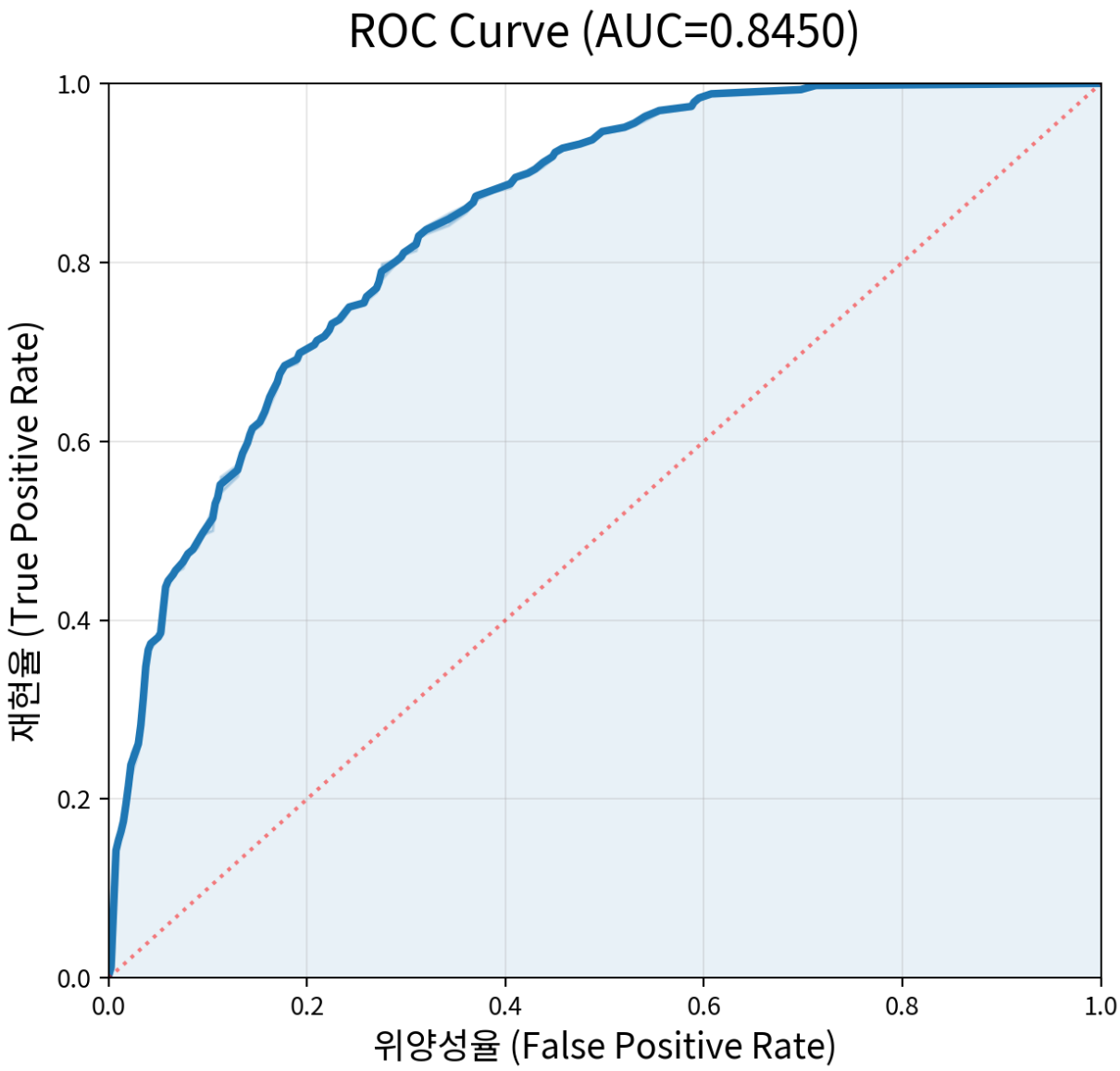
📘 #02. 로지스틱 회귀 모형

이 예제에서는 과적합 여부는 판정하지 않도록 한다. (실행시간 고려)

앞서 살펴봤지만 모든 모형 비교를 위해 다시 재현

```
%%time

logit_pipe = Pipeline([
    ("VIF_Selector", VIFSelector()),
```

```
    ("scaler", StandardScaler()),
    ("model", LogisticRegression(random_state=52))
])

logit_param_grid = {
    "model__penalty": ["l2"],
    "model__solver": ["lbfgs"],
    "model__C": [0.01, 0.1, 1, 10, 100],
    "model__max_iter": [100, 300, 500],
    "model__class_weight": [None, "balanced"],
}

logit_gs = GridSearchCV(
    estimator=logit_pipe,
    param_grid=logit_param_grid,
    cv=5,
    scoring="roc_auc",
    n_jobs=-1
)

logit_gs.fit(x_train, y_train)

logit_estimator = logit_gs.best_estimator_

logit_score_df = hs_cls_bin_scores(logit_estimator, x_test, y_test)
logit_score_df
```




```
CPU times: user 1.45 s, sys: 104 ms, total: 1.55 s
Wall time: 4.4 s
```

	정확도 (Accuracy)	정밀도 (Precision)	재현율 (Recall,tpr)	위양성율 (Fallout,fpr)	특이성 (TNR)	F1 Score	AUC
LogisticRegression	0.751	0.617	0.752	0.250	0.750	0.678	0.845

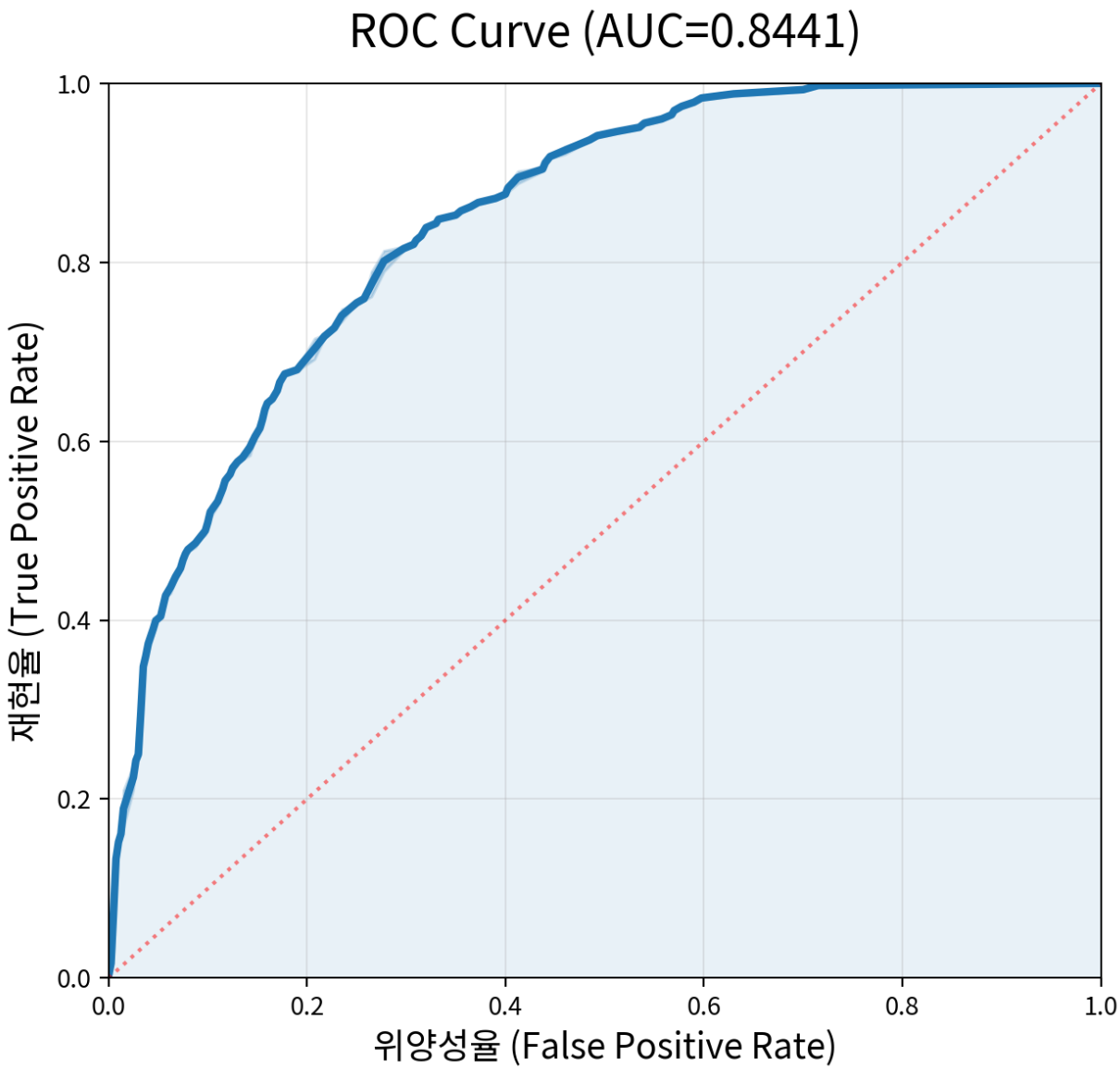
📘 #03. 경사하강법

하이퍼파라미터	구분	핵심도	의미	기본값 (sklearn)	역할	값 설정 가이드
`loss`	**분류**	⭐⭐⭐	손실함수	`'hinge'`	분류 기준 결정	• `'hinge'` → 선형 SVM • `'log_loss'` → 로지스틱 회귀 • `'modified_huber'` → 이상치 강건

penalty	공통	⭐⭐⭐	규제 종류	`'l2'`	적용할 규제 방식 선택	• `'l2'` → Ridge 성향 • `'l1'` → Lasso 성향 • `'elasticnet'` → 혼합
alpha	공통	⭐⭐⭐	규제 강도 (λ)	`0.0001`	계수 크기 억제 수준 결정	• 작게: 규제 약함 • 크게: 과적합 억제 강화
l1_ratio	공통	⭐⭐	L1 비율	`0.15`	elasticnet 사용 시 L1/L2 비율	• 0 → Ridge 성향 • 1 → Lasso 성향
max_iter	공통	⭐⭐	최대 반복 횟수	`1000`	학습 반복 한계	• 수렴 경고 시 증가
learning_rate	공통	⭐⭐	학습률 전략	회귀:'invscaling'`` 분류: `'optimal'`	학습률 감소 방식	• 기본값 유지 권장
eta0	공통	⭐	초기 학습률	`0.0`	초기 스텝 크기	• `'constant'`, `'adaptive'` 에서만 의미
class_weight	**분류**	⭐⭐	클래스 가중치	`None`	불균형 데이터 보정	• `'balanced'` 사용
random_state	공통	✅	난수 시드	`None`	학습 순서 재현성	• 실험 비교 시 고정
n_jobs	공통	✅	병렬 처리 수	`None`	연산 병렬화	• `-1` 권장

```
%%time

sgd_pipe = Pipeline([
    ("VIF_Selector", VIFSelector()),
    ("scaler", StandardScaler()),
```

```
    (
        "model",
        SGDClassifier(
            random_state=52,
            n_jobs=-1
        )
    ),
])

sgd_param_grid = {
    "model__loss": ["hinge", "log_loss", "modified_huber"],
    "model__penalty": ["l2", "l1", "elasticnet"],
    "model__alpha": [0.01, 0.1, 1, 10, 100],   # C의 역수 대응
    "model__max_iter": [1000, 2000, 5000],
    "model__class_weight": [None, "balanced"],
}

sgd_gs = GridSearchCV(
    estimator=sgd_pipe,
    param_grid=sgd_param_grid,
    cv=5,
    scoring="roc_auc",
    n_jobs=-1
)

sgd_gs.fit(x_train, y_train)

sgd_estimator = sgd_gs.best_estimator_

sgd_score_df = hs_cls_bin_scores(sgd_estimator, x_test, y_test)

sgd_score_df
```




```
CPU times: user 1.07 s, sys: 38.3 ms, total: 1.1 s
Wall time: 1.63 s
```

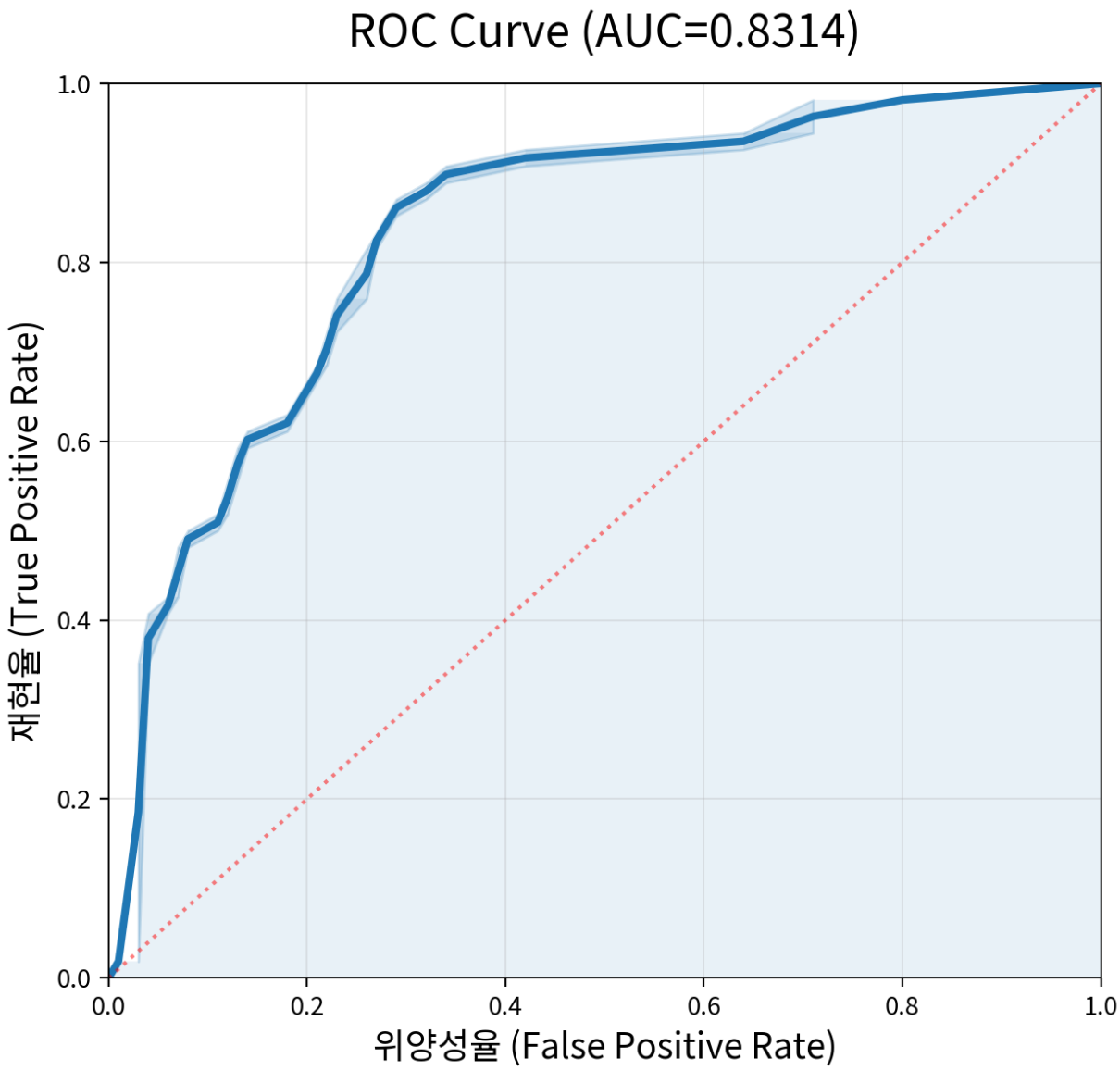
	정확도(Accuracy)	정밀도(Precision)	재현율(Recall,tpr)	위양성율(Fallout,fpr)	특이성(TNR)	F1 Score	AUC
SGDClassifier	0.769	0.709	0.570	0.125	0.875	0.632	0.844

📘 #04. KNN

파라미터명	구분	핵심도	설명	기본값	GridSearchCV 권장값
`n_neighbors`	공통	⭐⭐⭐	이웃 수 k (bias–variance 직접 제어)	`5`	`[3, 5, 7, 9, 15, 25]`
`weights`	공통	⭐⭐	이웃 가중치 방식	`'uniform'`	`['uniform', 'distance']`

metric	공통	⭐⭐	거리 계산 방식	`'minkowski'`	`['euclidean', 'manhattan', 'minkowski']`
p	공통	⭐⭐	Minkowski 거리 차수	`2`	`[1, 2]`
`algorithm`	공통	⭐	최근접 탐색 알고리즘	`'auto'`	`['auto', 'ball_tree', 'kd_tree']`
`leaf_size`	공통	⭐	트리 노드 크기	`30`	`[20, 30, 40]`
`n_jobs`	공통	✅	병렬 처리 수	`None`	`-1`
weights (확률)	분류	⭐⭐	클래스 확률 계산 영향	`'uniform'`	`'distance'` 권장

```
%%time

knn_cls_pipe = Pipeline([
    ("scaler", StandardScaler()),   # 거리기반 → 필수
    (
        "model",
        KNeighborsClassifier(n_jobs=-1)
    ),
])

knn_cls_param_grid = {
    "model__n_neighbors": [3, 5, 7, 9, 15, 25],
    "model__weights": ["uniform", "distance"],
    "model__metric": ["euclidean", "manhattan", "minkowski"],
    "model__p": [1, 2],
    "model__weights": ["uniform", "distance"],
}

knn_cls_gs = GridSearchCV(
    estimator=knn_cls_pipe,
    param_grid=knn_cls_param_grid,
    cv=5,
    scoring="roc_auc",
    n_jobs=-1
)

knn_cls_gs.fit(x_train, y_train)

knn_cls_estimator = knn_cls_gs.best_estimator_

knn_cls_score_df = hs_cls_bin_scores(
    knn_cls_estimator,
    x_test,
    y_test
)

knn_cls_score_df
```




```
CPU times: user 398 ms, sys: 14.5 ms, total: 413 ms
Wall time: 985 ms
```

	정확도 (Accuracy)	정밀도 (Precision)	재현율 (Recall,tpr)	위양성율 (Fallout,fpr)	특이성 (TNR)	F1 Score	AUC
KNeighborsClassifier	0.747	0.642	0.630	0.190	0.810	0.636	0.831

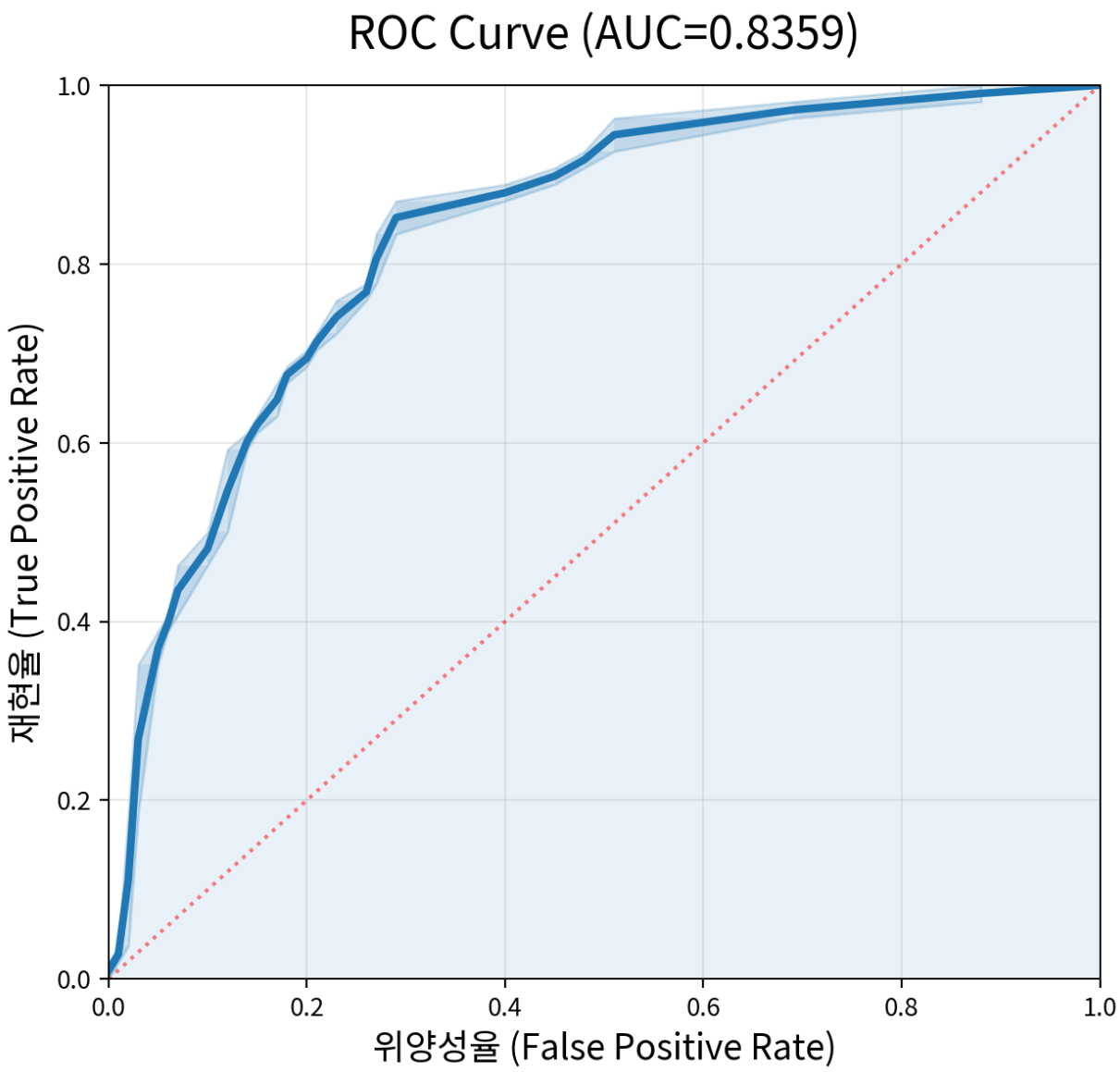
📘 #05. SVM

파라미터명	구분	핵심도	설명	기본값	GridSearchCV 권장값
`kernel`	공통	⭐⭐⭐	커널 종류 (비선형 표현 방식)	`'rbf'`	`['linear', 'rbf', 'poly']`
`C`	공통	⭐⭐⭐	오차 패널티 강도 (bias–variance 직접 제어)	`1.0`	`[0.1, 1, 10, 100]`
`epsilon`	예측	⭐⭐⭐	ε-튜브 폭 (오차 무시 범위)	`0.1`	`[0.01, 0.05, 0.1, 0.2, 0.5]`

gamma	공통	⭐⭐⭐	데이터 1개 영향 범위 (RBF/Poly)	'scale'	['scale', 'auto', 0.01, 0.1, 1]
degree	공통	⭐⭐	다항 커널 차수 (kernel='poly')	3	[2, 3, 4]
coef0	공통	⭐	다항·시그모이드 커널 보정항	0.0	[0.0, 0.5, 1.0]
shrinking	공통	⭐	최적화 속도 개선 옵션	True	[True, False]
tol	공통	⭐	수렴 허용 오차	1e-3	[1e-4, 1e-3]
max_iter	공통	✅	최대 반복 횟수	-1	-1
cache_size	공통	✅	커널 캐시 메모리(MB)	200	200
class_weight	분류	⭐⭐	클래스 불균형 보정	None	[None, 'balanced']

```
%%time

svc_pipe = Pipeline([
    ("scaler", StandardScaler()),   # SVM 필수
    (
        "model",
        SVC(
            random_state=52,
            cache_size=200,
            probability=True,   # 성능 평가를 위해 필수
        )
    ),
])

svc_param_grid = {
    "model__kernel": ["rbf"],   # 수업·실습 기본은 rbf 고정
    "model__C": [0.1, 1, 10, 100],
    "model__gamma": ["scale", "auto", 0.01, 0.1, 1],
    "model__degree": [2, 3, 4],                  # kernel='poly'에서만 의미
    "model__class_weight": [None, "balanced"],
}

svc_gs = GridSearchCV(
    estimator=svc_pipe,
    param_grid=svc_param_grid,
    cv=5,
    scoring="roc_auc",
    n_jobs=-1
)

svc_gs.fit(x_train, y_train)

svc_estimator = svc_gs.best_estimator_

svc_score_df = hs_cls_bin_scores(
    svc_estimator,
    x_test,
    y_test
)
```

```
svc_score_df
```




```
CPU times: user 770 ms, sys: 431 ms, total: 1.2 s
Wall time: 1.91 s
```

	정확도(Accuracy)	정밀도(Precision)	재현율(Recall,tpr)	위양성율(Fallout,fpr)	특이성(TNR)	F1 Score	AUC
SVC	0.753	0.608	0.833	0.290	0.710	0.703	0.836

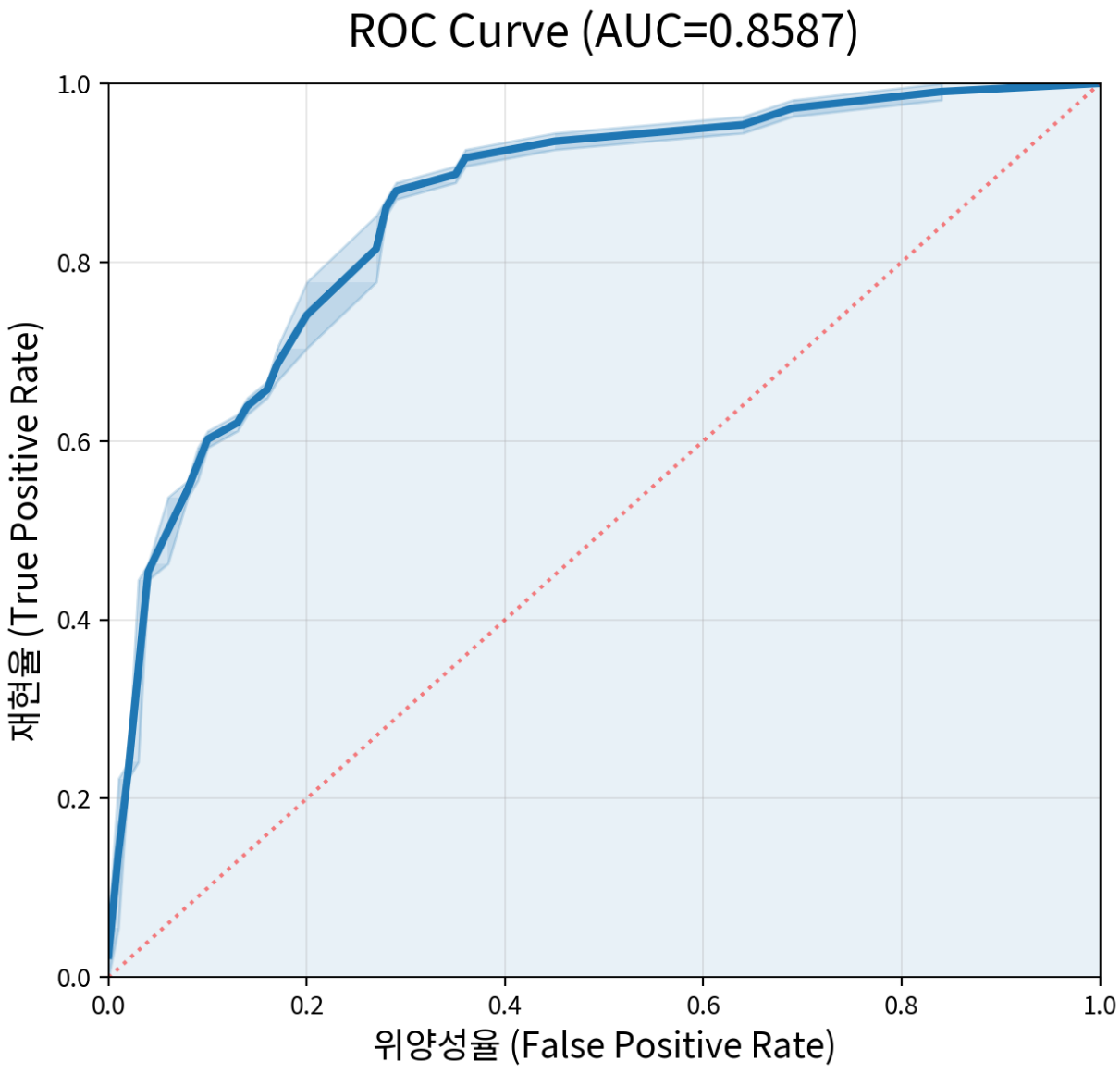
📘 #06. 랜덤 포레스트

파라미터명	구분	핵심도	실무 조정 전략 요약	기본값 (회귀 / 분류)	GridSearchCV 실무 권장값

n_estimators	공통	⭐⭐	성능 안정화용, 과도한 탐색 불필요	100	[300, 500]
max_depth	공통	⭐⭐	과적합 보조 제어	None	[None, 10]
min_samples_leaf	공통	⭐⭐⭐	과적합 통제 핵심	1	[5, 10]
max_features	공통	⭐⭐⭐	트리 다양성 핵심	회귀:1.0`` 분류: "sqrt"	['sqrt', 1.0]
criterion	구분별	⭐	분할 품질 기준	회귀:"squared_error"`` 분류: "gini"	회귀:["squared_error", "absolute_error"]`` 분류: ["gini", "entropy"]
class_weight	분류	⭐⭐	클래스 불균형 대응	None	[None, "balanced"]
n_jobs	공통	✅	병렬 처리	None	-1
random_state	공통	✅	재현성 확보	None	52

```
%%time

rf_cls_pipe = Pipeline([
    (
        "model",
        RandomForestClassifier(
            random_state=52,
            n_jobs=-1
        )
    ),
])

rf_cls_param_grid = {
    "model__n_estimators": [300, 500],
    "model__max_depth": [None, 10],
    "model__min_samples_leaf": [5, 10],
    "model__max_features": ["sqrt", 1.0],
    "model__criterion": ["gini", "entropy"],
    "model__class_weight": [None, "balanced"],
```

```
}

rf_cls_gs = GridSearchCV(
    estimator=rf_cls_pipe,
    param_grid=rf_cls_param_grid,
    cv=5,
    scoring="roc_auc",
    n_jobs=-1
)

rf_cls_gs.fit(x_train, y_train)

rf_cls_estimator = rf_cls_gs.best_estimator_

rf_cls_score_df = hs_cls_bin_scores(
    rf_cls_estimator,
    x_test,
    y_test
)

rf_cls_score_df
```



```
CPU times: user 1.12 s, sys: 798 ms, total: 1.92 s
Wall time: 18.3 s
```

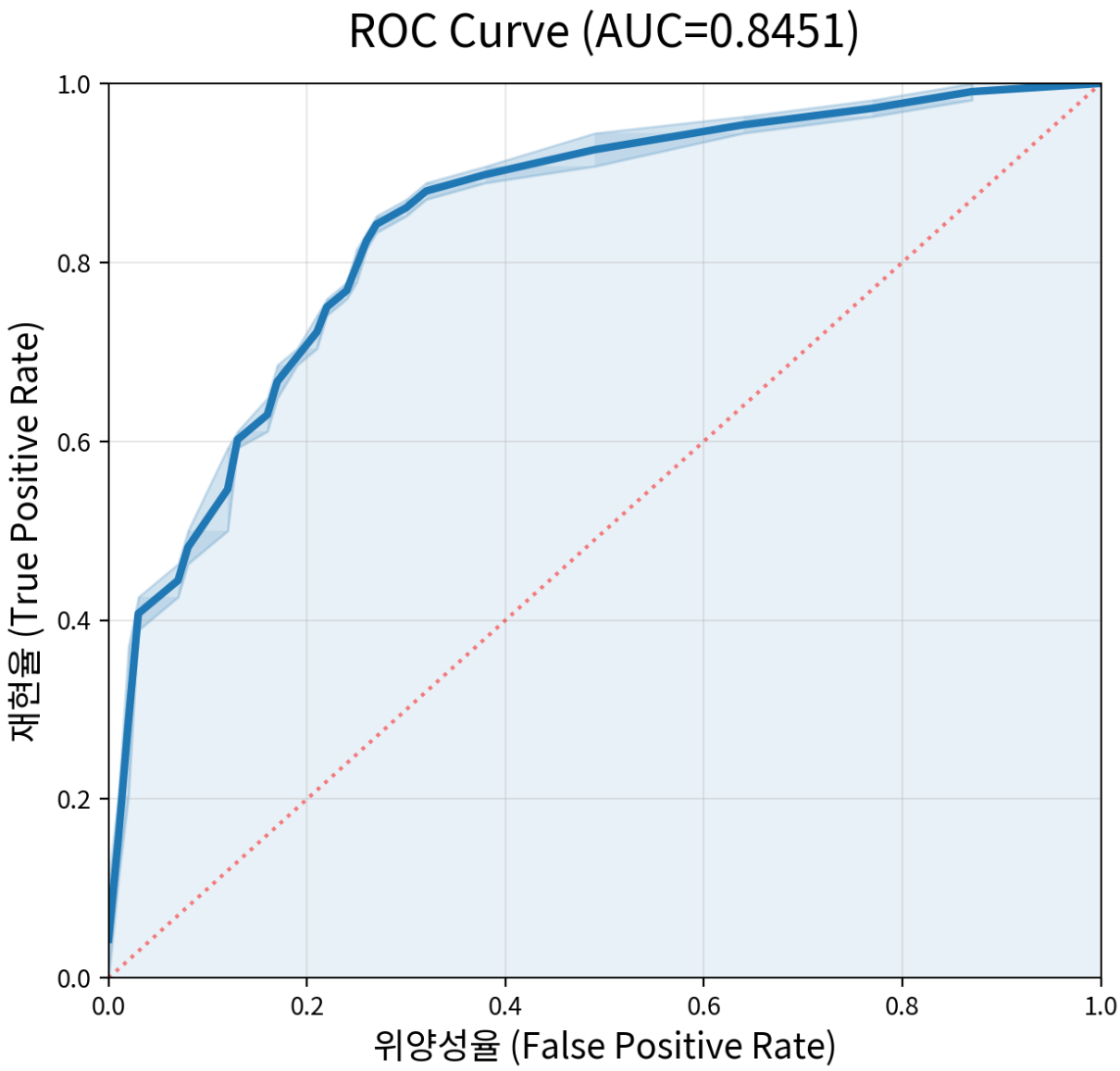
	정확도 (Accuracy)	정밀도 (Precision)	재현율 (Recall,tpr)	위양성율 (Fallout,fpr)	특이성 (TNR)	F1 Score	AUC
RandomForestClassifier	0.747	0.609	0.778	0.270	0.730	0.683	0.859

📘 #07. XGBoost

파라미터명	구분	핵심도	설명	기본값 (회귀 / 분류)	GridSearchCV 권장값
n_estimators	공통	⭐⭐⭐	부스팅 트리 개수	100	[100, 300, 500, 800]
learning_rate	공통	⭐⭐⭐	각 트리 기여도	0.3	[0.01, 0.05, 0.1, 0.2, 0.3]
max_depth	공통	⭐⭐⭐	개별 트리 깊이	6	[3, 5, 7, 10]
min_child_weight	공통	⭐⭐⭐	리프 분기 최소 가중치	1	[1, 5, 10, 20]
gamma	공통	⭐⭐	분기 최소 손실 감소량	0	[0, 0.1, 0.5, 1, 5]
subsample	공통	⭐⭐	행 샘플링 비율	1.0	[0.6, 0.8, 1.0]
colsample_bytree	공통	⭐⭐	열 샘플링 비율	1.0	[0.6, 0.8, 1.0]
reg_alpha	공통	⭐⭐	L1 정규화	0	[0, 0.1, 1, 10]
reg_lambda	공통	⭐⭐	L2 정규화	1	[1, 5, 10, 50]
objective	구분별	✅	손실 함수	회귀:reg:squarederror`` 분류: `binary:logistic`	고정
scale_pos_weight	분류	⭐⭐	클래스 불	1	[1, 2, 5, 10]

			균형 보정		
random_state	공통	✅	재현성 확보	None	52
n_jobs	공통	✅	병렬 처리	-1	-1

```
%%time

xgb_cls_pipe = Pipeline([
    (
        "model",
        XGBClassifier(
            objective="binary:logistic",
            random_state=52,
            n_jobs=-1
        )
    ),
])

xgb_cls_param_grid = {
    "model__n_estimators": [100, 300],
    "model__learning_rate": [0.01, 0.05, 0.1],
    "model__max_depth": [3, 5, 7],
    "model__min_child_weight": [1, 5, 10],
    "model__subsample": [0.6, 0.8, 1.0],
    "model__colsample_bytree": [0.6, 0.8, 1.0],
    "model__reg_alpha": [0, 0.1, 1],
    "model__reg_lambda": [1, 5, 10],
    "model__scale_pos_weight": [1],  # 불균형 시 조정
}

xgb_cls_gs = GridSearchCV(
    estimator=xgb_cls_pipe,
    param_grid=xgb_cls_param_grid,
    cv=5,
    scoring="roc_auc",
    n_jobs=-1
)

xgb_cls_gs.fit(x_train, y_train)

xgb_cls_estimator = xgb_cls_gs.best_estimator_

xgb_cls_score_df = hs_cls_bin_scores(
    xgb_cls_estimator,
    x_test,
    y_test
)

xgb_cls_score_df
```




```
CPU times: user 9.86 s, sys: 851 ms, total: 10.7 s
Wall time: 1min 8s
```

	정확도(Accuracy)	정밀도(Precision)	재현율(Recall,tpr)	위양성율(Fallout,fpr)	특이성(TNR)	F1 Score	AUC
XGBClassifier	0.766	0.680	0.630	0.160	0.840	0.654	0.845

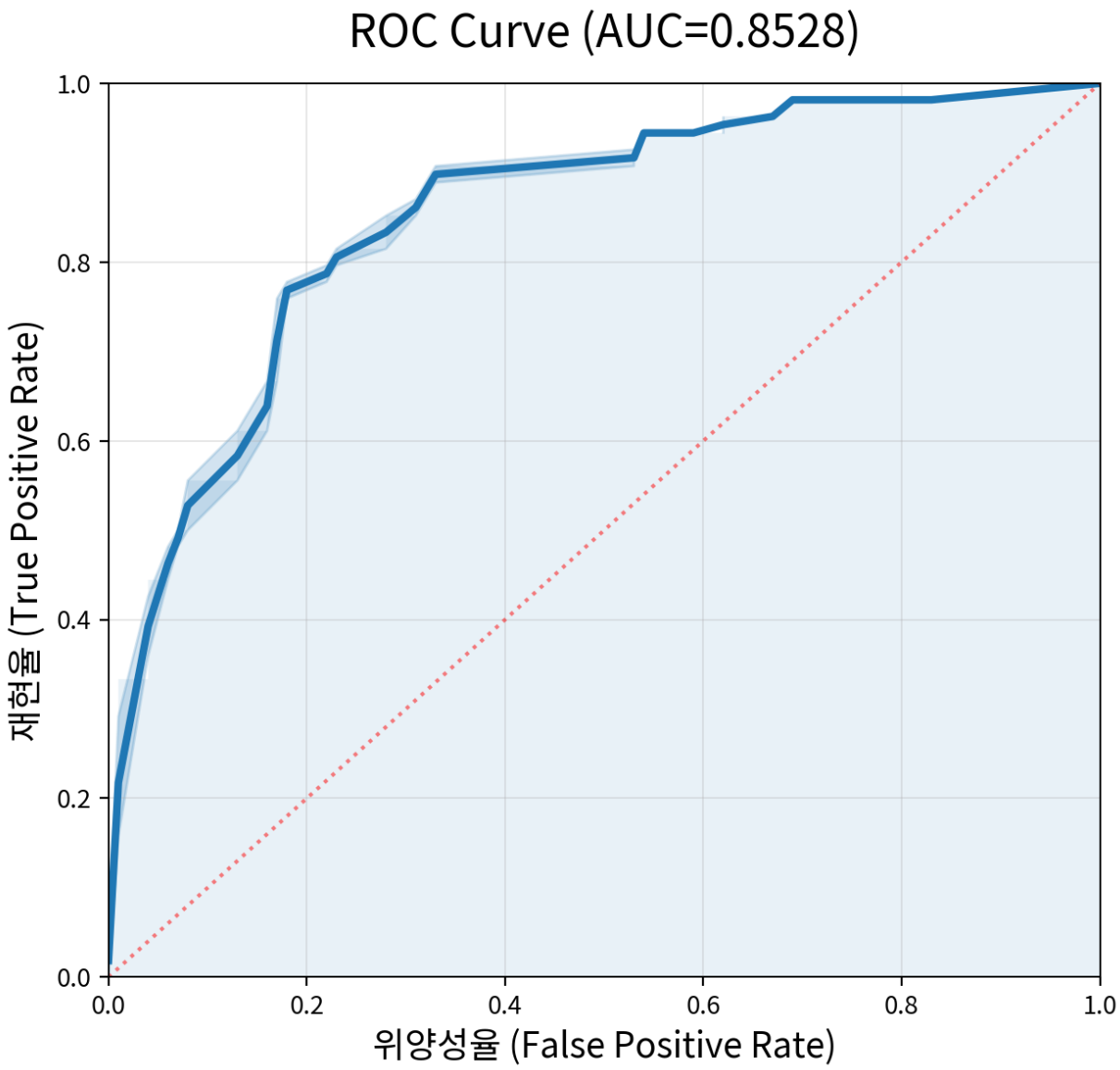
📘 #08. LightGBM

파라미터명	구분	핵심도	설명	기본값 (회귀 / 분류)	GridSearchCV 권장값
`n_estimators`	공통	⭐⭐⭐	부스팅 트리 개수	`100`	`[100, 300, 500, 800]`
`learning_rate`	공통	⭐⭐⭐	각 트리 기여도	`0.1`	`[0.01, 0.05, 0.1]`
`num_leaves`		⭐⭐⭐		`31`	`[15, 31, 63]`

	공통		리프 개수 (복잡도 핵심)		
max_depth	공통	⭐⭐⭐	최대 깊이 (leaf-wise 과적합 제어)	-1	[-1, 3, 5, 7]
min_child_samples	공통	⭐⭐⭐	리프 최소 샘플 수	20	[10, 20, 50, 100]
min_child_weight	공통	⭐⭐	리프 최소 가중치 합	1e-3	[1e-3, 1e-2, 1e-1]
subsample	공통	⭐⭐	행 샘플링 비율	1.0	[0.6, 0.8, 1.0]
subsample_freq	공통	⭐⭐	subsample 적용 빈도	0	[0, 1]
reg_alpha	공통	⭐⭐	L1 정규화	0.0	[0, 0.1, 1]
reg_lambda	공통	⭐⭐	L2 정규화	0.0	[0, 1, 5, 10]
objective	구분별	⭐	손실 함수	회귀:regression`` 분류:binary`` 다중분류: multiclass	고정
random_state	공통	✅	재현성 시드	None	52
n_jobs	공통	✅	병렬 처리	-1	-1

```
%%time

lgbm_cls_pipe = Pipeline([
    (
        "model",
        LGBMClassifier(
            objective="binary",
            random_state=52,
            n_jobs=-1,
            verbose=-1
        )
    ),
])

lgbm_cls_param_grid = {
    "model__n_estimators": [100, 300],
    "model__learning_rate": [0.01, 0.05, 0.1],
    # "model__num_leaves": [15, 31],
    # "model__max_depth": [3, 5],
    # "model__min_child_samples": [10, 20],
    # "model__subsample": [0.6, 0.8, 1.0],
    "model__reg_alpha": [0, 0.1, 1],
```

```
    "model__reg_lambda": [0, 1, 5],
}

lgbm_cls_gs = GridSearchCV(
    estimator=lgbm_cls_pipe,
    param_grid=lgbm_cls_param_grid,
    cv=5,
    scoring="roc_auc",
    n_jobs=-1
)

lgbm_cls_gs.fit(x_train, y_train)

lgbm_cls_estimator = lgbm_cls_gs.best_estimator_

lgbm_cls_score_df = hs_cls_bin_scores(
    lgbm_cls_estimator,
    x_test,
    y_test
)

lgbm_cls_score_df
```



```
CPU times: user 743 ms, sys: 1.26 s, total: 2.01 s
Wall time: 1min 1s
```

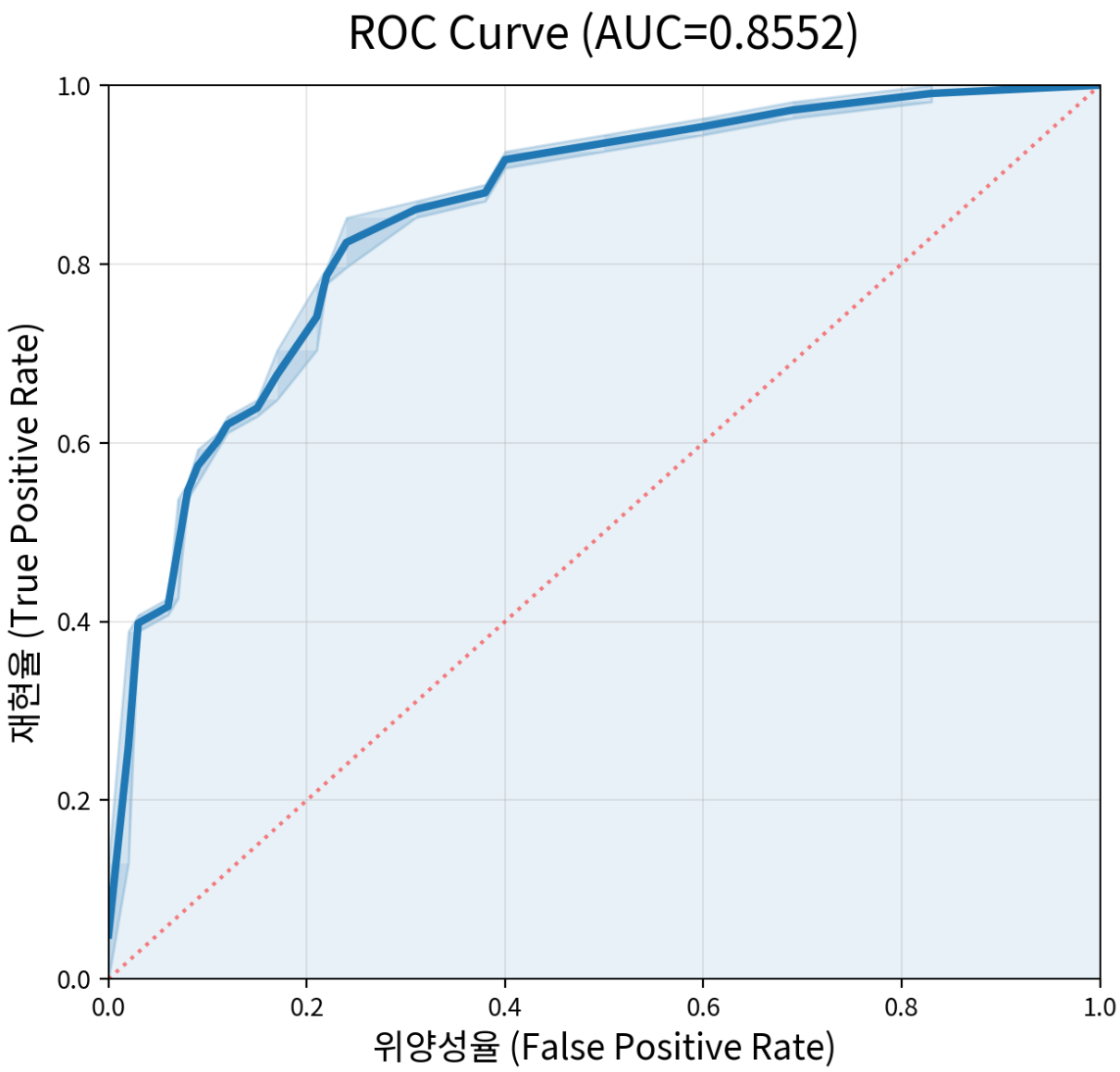
	정확도 (Accuracy)	정밀도 (Precision)	재현율 (Recall,tpr)	위양성율 (Fallout,fpr)	특이성 (TNR)	F1 Score	AUC
LGBMClassifier	0.766	0.705	0.574	0.130	0.870	0.633	0.853

📘 #09. CatBoost

파라미터명	구분	핵심도	설명	기본값 (회귀 / 분류)	GridSearchCV 권장값
iterations	공통	⭐⭐⭐	부스팅 트리 개수	1000	[300, 500, 800, 1000]
learning_rate	공통	⭐⭐⭐	각 트리 학습률	0.03	[0.01, 0.03, 0.1]
depth	공통	⭐⭐⭐	대칭 트리 깊이	6	[4, 6, 8, 10]
l2_leaf_reg	공통	⭐⭐⭐	리프 L2 정규화	3.0	[1, 3, 5, 10]
min_data_in_leaf	공통	⭐⭐	리프 최소 샘플 수	1	[1, 10, 30, 50]
subsample	공통	⭐⭐	행 샘플링 비율	1.0	[0.6, 0.8, 1.0]
rsm	공통	⭐⭐	열 샘플링 비율	1.0	[0.6, 0.8, 1.0]
loss_function	구분별	⭐⭐	손실 함수	회귀:RMSE`` 분류: Logloss	고정
eval_metric	구분별	⭐	평가 지표	회귀:RMSE`` 분류: AUC	고정
bootstrap_type	공통	⭐⭐	샘플링 방식	'Bayesian'	['Bayesian', 'Bernoulli']
bagging_temperature	공통	⭐⭐	Bayesian bootstrap 강도	1.0	[0, 1, 5]
random_strength	공통	⭐⭐	분기 랜덤 노이즈	1.0	[0, 1, 5]
early_stopping_rounds	공통	⭐⭐	조기 종료	None	[50, 100]
cat_features	공통	⭐⭐⭐	범주형 변수 지정	None	고정 (데이터 의존)
random_state	공통	✅	재현성 시드	None	52
thread_count	공통	✅	병렬 처리	-1	-1

```
%%time

cat_cls_pipe = Pipeline([
    (
        "model",
        CatBoostClassifier(
            loss_function="Logloss",
            eval_metric="AUC",
            cat_features=[],  # 범주형 변수 컬럼명 또는 인덱스 지정
            random_state=52,
            thread_count=-1,
            verbose=0
        )
    ),
])

cat_cls_param_grid = [
    {   # Bayesian → subsample 제거
        "model__bootstrap_type": ["Bayesian"],
        "model__iterations": [300, 500],
        "model__learning_rate": [0.01, 0.03],
        "model__depth": [4, 6],
        "model__l2_leaf_reg": [1, 3],
        "model__rsm": [0.6, 1.0],
    },
    {   # Bernoulli → subsample 허용
        "model__bootstrap_type": ["Bernoulli"],
        "model__iterations": [300, 500],
        "model__learning_rate": [0.01, 0.03],
        "model__depth": [4, 6],
        "model__l2_leaf_reg": [1, 3],
        "model__subsample": [0.6, 1.0],
        "model__rsm": [0.6, 1.0],
    },
]

cat_cls_gs = GridSearchCV(
    estimator=cat_cls_pipe,
    param_grid=cat_cls_param_grid,
    cv=5,
    scoring="roc_auc",
    n_jobs=-1
)

cat_cls_gs.fit(
    x_train,
    y_train
)

cat_cls_estimator = cat_cls_gs.best_estimator_

cat_cls_score_df = hs_cls_bin_scores(
    cat_cls_estimator,
    x_test,
    y_test
)
```

```
cat_cls_score_df
```




```
CPU times: user 1.07 s, sys: 137 ms, total: 1.21 s
Wall time: 12.3 s
```

	정확도 (Accuracy)	정밀도 (Precision)	재현율 (Recall,tpr)	위양성율 (Fallout,fpr)	특이성 (TNR)	F1 Score	AUC
CatBoostClassifier	0.786	0.733	0.611	0.120	0.880	0.667	0.855

📘 #10. 결과 정리

📝 AUC를 우선으로 고려해야 하는 경우

누가 더 위험한지 순위를 매기는 문제

- 임계값이 아직 정해지지 않은 문제
  - 신용 위험 점수 모델
  - 질병 위험 예측
  - 고객 이탈 확률 예측
- 클래스 불균형이 큰 문제
  - 사기 탐지 (0.1%)
  - 고장 탐지
- 비용이 상황에 따라 변하는 문제
  - 마케팅 캠페인 타겟 선정
  - 보험 리스크 평가

```
result_df = concat([logit_score_df, sgd_score_df, knn_cls_score_df, svc_score_df,
        rf_cls_score_df, xgb_cls_score_df, lgbm_cls_score_df, cat_cls_score_df])

result_df.sort_values('AUC', ascending=False, inplace=True)

result_df
```

	정확도 (Accuracy)	정밀도 (Precision)	재현율 (Recall,tpr)	위양성율 (Fallout,fpr)	특이성 (TNR)	F1 Score	AUC
RandomForestClassifier	0.747	0.609	0.778	0.270	0.730	0.683	0.859
CatBoostClassifier	0.786	0.733	0.611	0.120	0.880	0.667	0.855
LGBMClassifier	0.766	0.705	0.574	0.130	0.870	0.633	0.853
XGBClassifier	0.766	0.680	0.630	0.160	0.840	0.654	0.845
LogisticRegression	0.751	0.617	0.752	0.250	0.750	0.678	0.845
SGDClassifier	0.769	0.709	0.570	0.125	0.875	0.632	0.844
SVC	0.753	0.608	0.833	0.290	0.710	0.703	0.836
KNeighborsClassifier	0.747	0.642	0.630	0.190	0.810	0.636	0.831

📝 Recall을 우선 고려해야 하는 경우

놓치면 안 되는 대상을 반드시 찾아내야 하는 문제
(FN, False Negative 비용이 매우 큰 상황)

- 생명·안전과 직결되는 문제
  - 암 진단

    - 중증 질환 조기 발견
    - 감염병 선별 검사
- 재난·사고 예방 문제
    - 산업 설비 고장 탐지
    - 화재·가스 누출 탐지
    - 자율주행 보행자 인식
- 보안·위험 탐지 문제
    - 침입 탐지 시스템
    - 이상 거래 탐지 (사후 차단 가능할 때)
    - 테러·위험 물질 탐지
- 1차 스크리닝 단계 문제
    - 건강검진 선별 모델
    - 대출 심사 1차 위험군 분류
    - 보험 리스크 1차 분류

```
result_df.sort_values('재현율(Recall,tpr)', ascending=False)
```

	정확도 (Accuracy)	정밀도 (Precision)	재현율 (Recall,tpr)	위양성율 (Fallout,fpr)	특이성 (TNR)	F1 Score	AUC
SVC	0.753	0.608	0.833	0.290	0.710	0.703	0.836
RandomForestClassifier	0.747	0.609	0.778	0.270	0.730	0.683	0.859
LogisticRegression	0.751	0.617	0.752	0.250	0.750	0.678	0.845
XGBClassifier	0.766	0.680	0.630	0.160	0.840	0.654	0.845
KNeighborsClassifier	0.747	0.642	0.630	0.190	0.810	0.636	0.831
CatBoostClassifier	0.786	0.733	0.611	0.120	0.880	0.667	0.855
LGBMClassifier	0.766	0.705	0.574	0.130	0.870	0.633	0.853
SGDClassifier	0.769	0.709	0.570	0.125	0.875	0.632	0.844

📝 Accuracy(정확도)를 우선 고려해야 하는 경우

양성과 음성의 비율이 균형에 가깝고, 오분류 비용이 유사한 문제

- 일반적인 다중분류 문제
    - 이미지 분류
    - 문서 주제 분류
    - 제품 카테고리 분류
- 오분류 비용이 거의 동일한 문제
    - 간단한 품질 등급 분류
    - 고객 세그먼트 분류
- 불균형이 크지 않은 데이터셋
    - 클래스 비율이 40:60 수준

> 핵심 기준:
> “틀리는 것이 크게 위험하지 않고, 전체적으로 얼마나 맞추는가가 중요한가?”

📝 Precision(정밀도)를 우선 고려해야 하는 경우

양성으로 예측한 것의 신뢰도가 중요한 문제
(FP, False Positive 비용이 큰 상황)

- 의료 진단 확정 단계
  - 고위험 수술 대상 판정
  - 강한 부작용 치료 적용 대상 선정
- 법적·재정적 책임이 큰 문제
  - 보험 사기 확정 판정
  - 범죄 혐의 분류
- 마케팅 비용이 높은 타겟팅
  - 고가 상품 캠페인
  - VIP 전용 프로모션

> 핵심 기준:
> “양성이라고 판단했는데 실제로 음성이면 큰 비용이 발생하는가?”

📝 Fallout (FPR, 위양성율)를 우선 고려해야 하는 경우

정상 집단을 잘못 양성으로 분류하면 큰 손실이 발생하는 문제

- 정상 고객 차단 위험
  - 카드 결제 차단
  - 계정 정지 시스템
- 정상 환자 오진
  - 불필요한 검사·치료
- 서비스 품질 유지 문제
  - 정상 사용자의 이용 제한

📝 Specificity (특이성, TNR)를 우선 고려해야 하는 경우

정상(음성)을 정확히 유지하는 것이 중요한 문제

- 대규모 스크리닝 검사
  - 정상자를 최대한 정상으로 유지
- 자동 승인 시스템
  - 정상 대출 승인
- 고객 유지 전략
  - 정상 고객을 이탈 위험군으로 오분류 방지

📝 F1 Score를 우선 고려해야 하는 경우

Precision과 Recall을 동시에 고려해야 하는 불균형 문제

- 사기 탐지
- 의료 진단 중간 단계
- 이상 탐지 모델 평가