Chapter 9. Practice: Metric, CV, Optimization

SKLearn - Metric

- Classification
  - accuracy_score, f1_score, roc_auc_score, average_precision_score
- Regression
  - mean_squared_error, mean_absolute_percentage_error, r2_score

SKLearn - Metric

- Accuracy and F1 score: 예측 결과 필요
- AUROC, PRAUC : 예측 확률 필요
- Sklearn 모델의 경우 predict( )는 분류 결과, predict_proba( )는 분류 확률 출력

Metric with Hold-out validation

- train_test_split 함수 사용
-
```
!wget https://raw.githubusercontent.com/shryu8902/KIRD_AUTOML/main/Titanic-Dataset.csv
import pandas as pd
df = pd.read_csv('./Titanic-Dataset.csv')
```

Preprocessing

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

mean_imputer = SimpleImputer(strategy='mean')
df['Age'] = mean_imputer.fit_transform(df[['Age']])

label_encoder = LabelEncoder()
df['Sex'] = label_encoder.fit_transform(df['Sex'])
df['Pclass'] = label_encoder.fit_transform(df['Pclass'])

mode_imputer = SimpleImputer(strategy='most_frequent')
df[['Embarked']] = mode_imputer.fit_transform(df[['Embarked']])
df['Embarked'] = label_encoder.fit_transform(df['Embarked'])
```

Train-test split

```
from sklearn.model_selection import train_test_split
X_, X_test, y_, y_test = train_test_split(df_X, df_y, test_size = 0.4, random_state = 42)
X_train, X_val, y_train, y_val = train_test_split(X_,y_, test_size = 0.5, random_state = 42)
print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```

Model training

```
from sklearn.ensemble import RandomForestClassifier

model_1 = RandomForestClassifier()
model_1.fit(X_train, y_train)

y_pred_1 = model_1.predict(X_val)
y_pred_proba_1 = model_1.predict_proba(X_val)
```

```
from sklearn.linear_model import LogisticRegression

model_2 = LogisticRegression()
model_2.fit(X_train, y_train)

y_pred_2 = model_2.predict(X_val)
y_pred_proba_2 = model_2.predict_proba(X_val)
```

Error calculation - validation

```
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, average_precision_score

#accuracy
print(accuracy_score(y_val, y_pred_1))
print(accuracy_score(y_val, y_pred_2))

#f1_score
print(f1_score(y_val, y_pred_1))
print(f1_score(y_val, y_pred_2))

#AUROC
print(roc_auc_score(y_val, y_pred_proba_1[:,1]))
print(roc_auc_score(y_val, y_pred_proba_2[:,1]))

#PRAUC
print(average_precision_score(y_val, y_pred_proba_1[:,1]))
print(average_precision_score(y_val, y_pred_proba_2[:,1]))
```

Error calculation - test

```
y_pred_test1 = model_1.predict(X_test)
y_pred_proba_test1 = model_1.predict_proba(X_test)
y_pred_test2 = model_2.predict(X_test)
y_pred_proba_test2 = model_2.predict_proba(X_test)

#accuracy
print(accuracy_score(y_test, y_pred_test1))
print(accuracy_score(y_test, y_pred_test2),'\n')

#f1_score
print(f1_score(y_test, y_pred_test1))
print(f1_score(y_test, y_pred_test2),'\n')

#AUROC
print(roc_auc_score(y_test, y_pred_proba_test1[:,1]))
print(roc_auc_score(y_test, y_pred_proba_test2[:,1]),'\n')

#PRAUC
print(average_precision_score(y_test, y_pred_proba_test1[:,1]))
print(average_precision_score(y_test, y_pred_proba_test2[:,1]))
```

오차 비교

	Validation	Test
Accuracy	0.80 0.81	0.80 0.79
F1 score	0.70 0.71	0.74 0.72
AUROC	0.83 0.83	0.86 0.85
PRAUC	0.79 0.79	0.79 0.84

Cross validation - K fold

```
from sklearn.model_selection import KFold
import numpy as np

kf = KFold(n_splits = 5, shuffle = True, random_state = 42)

acc_per_fold = []

for train_index, val_index in kf.split(X_):
    X_train, X_val = X_.iloc[train_index], X_.iloc[val_index]
    y_train, y_val = y_.iloc[train_index], y_.iloc[val_index]

    model = RandomForestClassifier(random_state = 0)
    model.fit(X_train, y_train)

    model_pred = model.predict(X_val)
    model_pred_proba = model.predict_proba(X_val)

    acc = accuracy_score(y_val, model_pred)
    acc_per_fold.append(acc)

kf_val_acc = np.mean(acc_per_fold)
print(kf_val_acc)
```

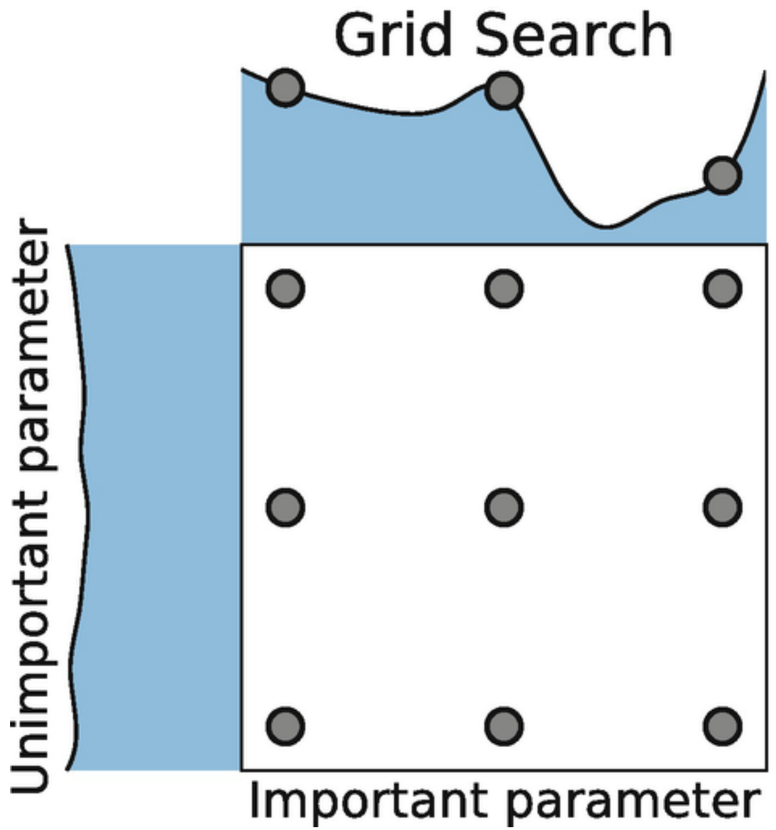
Cross Validation

```
from sklearn.model_selection import cross_val_score

model = RandomForestClassifier(random_state = 0)
cv_scores = cross_val_score(model, X_, y_, cv = 5)
print(cv_scores)
print(np.mean(cv_scores))
```

Optimization - Grid search

```
class sklearn.model_selection.GridSearchCV(estimator, param_grid, *,
scoring=None, n_jobs=None, refit=True, cv=None, verbose=0,
pre_dispatch='2*n_jobs', error_score=nan, return_train_score=False)
```

- Estimator : ML model
- Param_grid : 탐색 공간을 정의하는 dictionary
- Dict : { “key”:value}
- Refit : best_params으로 학습데이터 재학습



Optimization - Random Forest

RandomForestClassifier

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *,
criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None,
min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None,
random_state=None, verbose=0, warm_start=False, class_weight=None,
ccp_alpha=0.0, max_samples=None, monotonic_cst=None) #
```

[source]

- n_estimators = 1, 10, 100
- Criterion = “gini”, “entropy”, “log_loss”
- max_depth = None, 2, 5

```
param_grid = {
    'n_estimators': [1, 10, 100],
    'criterion': ['gini', 'entropy', 'log_loss'],
    'max_depth': [None, 2, 5]
}
```

```python
from sklearn.model_selection import GridSearchCV

#estimator 객체
model = RandomForestClassifier(random_state=0)

param_grid = {
    'n_estimators': [1, 10, 100],
    'criterion': ['gini', 'entropy', 'log_loss'],
    'max_depth': [None, 2, 5]
}

gridcv = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

gridcv.fit(X_, y_)

print("Best params:", gridcv.best_params_)
print("Best score:", gridcv.best_score_)
```

Gridsearch CV - best model

```
y_pred = gridcv.best_estimator_.predict(X_test)
y_pred_proba = gridcv.best_estimator_.predict_proba(X_test)

#accuracy
print(accuracy_score(y_test, y_pred),'\n')

#f1_score
print(f1_score(y_test, y_pred),'\n')

#AUROC
print(roc_auc_score(y_test, y_pred_proba[:,1]),'\n')

#PRAUC
print(average_precision_score(y_test, y_pred_proba[:,1]),'\n')
```

Optimization - random search

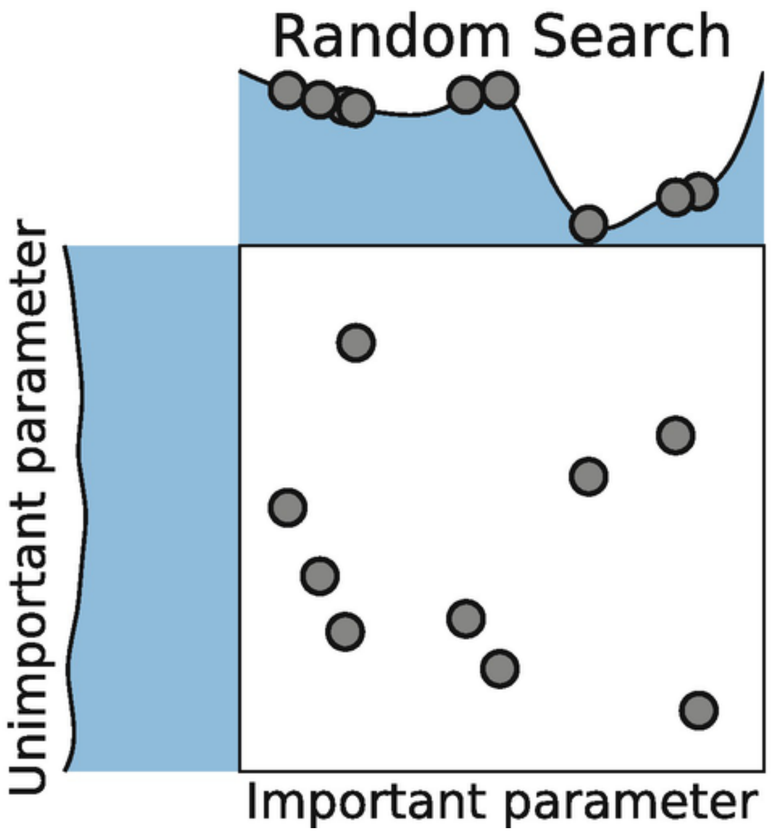
RandomizedSearchCV

```
class sklearn.model_selection.RandomizedSearchCV(estimator,
param_distributions, *, n_iter=10, scoring=None, n_jobs=None, refit=True,
cv=None, verbose=0, pre_dispatch='2*n_jobs', random_state=None,
error_score=nan, return_train_score=False)
```

[source]

- Param_distributions :
  탐색 공간을 정의하는 dictionary.
  확률 분포에 기반

- n_iter : 랜덤 조합 테스트 횟수



```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

# 하이퍼파라미터 탐색 공간 (범위 지정)
param_dist = {
    'n_estimators': randint(10, 201),    # 10 ~ 200 사이 정수
    'max_depth': randint(2, 11),         # 2 ~ 10 사이 정수
    'criterion': ['gini', 'entropy', 'log_loss']
}

#estimator 객체
model = RandomForestClassifier(random_state=0)

randcv = RandomizedSearchCV(
    estimator=model,
    param_distributions=param_grid,
    n_iter = 50,
    cv=5,
    scoring='accuracy',
)

randcv.fit(X_, y_)

print("Best params:", randcv.best_params_)
print("Best score:", randcv.best_score_)
```

Randomsearch CV - best model

```
y_pred = randcv.best_estimator_.predict(X_test)
y_pred_proba = randcv.best_estimator_.predict_proba(X_test)

#accuracy
print(accuracy_score(y_test, y_pred),'\n')

#f1_score
print(f1_score(y_test, y_pred),'\n')

#AUROC
print(roc_auc_score(y_test, y_pred_proba[:,1]),'\n')

#PRAUC
print(average_precision_score(y_test, y_pred_proba[:,1]),'\n')
```

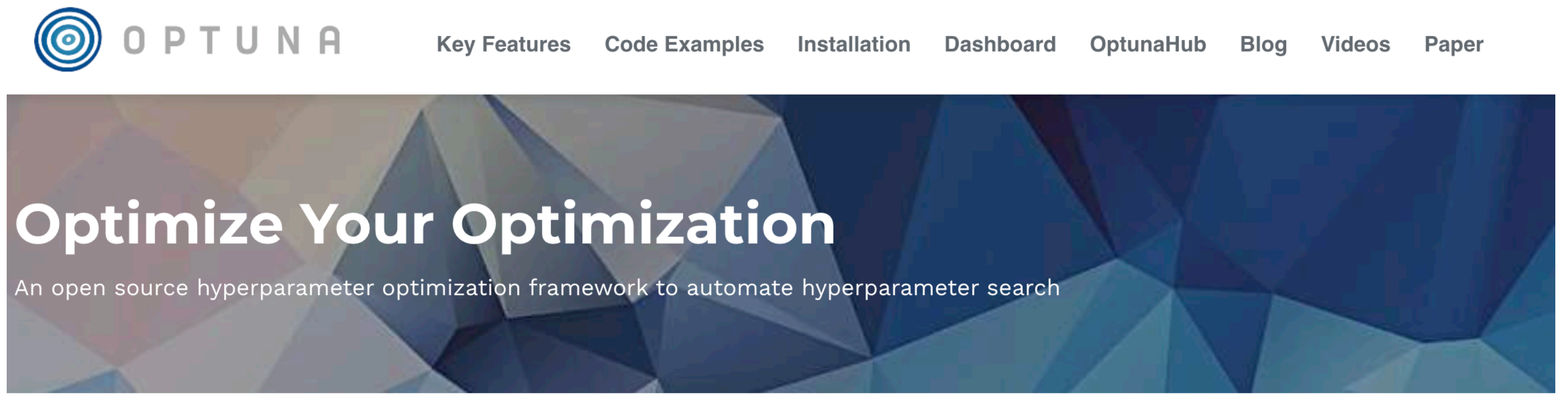
Basic lightgbm

```python
import lightgbm as lgb

# LightGBM 모델 설정
params = {
    'objective': 'binary',
    'metric': 'binary_logloss',
    'boosting_type': 'gbdt',
    'max_depth' : 3,
    'learning_rate': 0.1,
}

# LightGBM 모델 학습
model = lgb.LGBMClassifier(**params)
model.fit(X_train, y_train)

# 테스트 데이터에 대한 예측
y_pred = model.predict(X_test)

# 정확도 평가
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

LGBM - without validation

	Test	Gridsearch cv	Randomsearch cv (n_iter = 50)	LGBM
Accuracy	0.80 0.79	0.81	0.80	0.81
F1 score	0.74 0.72	0.73	0.72	0.75
AUROC	0.86 0.85	0.85	0.85	0.85
PRAUC	0.79 0.84	0.83	0.84	0.80

Optimization with optuna



Optimization with optuna

- Trial: Trial은 단일 최적화 시도를 나타내는 객체입니다.
  - Trial 객체는 하나의 하이퍼파라미터 조합과 해당 조합에 대한 목적 함수의 평가 결과를 저장
- Study: Study는 Optuna에서 전체 최적화 과정을 관리하는 객체
  - 하이퍼파라미터 조합에 대한 성능 추적과 최적화 알고리즘의 실행을 담당
  - 하나의 Study 객체는 하나의 최적화 작업을 의미하고 여러 개의 Trial 객체를 생성하여 하이퍼파라미터 조합을 탐색하고 평가합니다.
- Trial Suggest: Trial Suggest는 하이퍼파라미터 값을 추천하는 메서드를 제공하는 객체
  - suggest_uniform, suggest_categorical, suggest_loguniform 등의 메서드를 사용하여 연속형, 범주형, 로그 스케일 등의 하이퍼파라미터 값을 추천 가능.
  - Objective Function 내에서 Trial Suggest를 호출하여 추천된 하이퍼파라미터 값을 가져올 수 있습니다.

Optuna 기본예제

```python
import optuna
# 기본적인 예제
def objective(trial):
    x = trial.suggest_float("x", 0, 10)
    return (x-2)**2

study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=10)
print(study.best_params)
```

- n_trial 횟수를 늘리면?

Objective 설정

```python
# 목적 함수 구성.
# 목적 함수는 최적화 대상 값을 반환해야합니다. e.g., 점수, 정확도, 로스 등

def objective(trial):
    # 탐색을 위한 범위 설정
    params = {
        'objective': 'binary',
        'metric': 'binary_logloss',
        'boosting_type': trial.suggest_categorical('boosting_type', ['gbdt', 'dart', 'goss']),
        'num_leaves': trial.suggest_int('num_leaves', 10, 100),
        'max_depth' : trial.suggest_int('max_depth', 1, 10),
        'learning_rate': trial.suggest_loguniform('learning_rate', 0.01, 0.1),
    }
    # 최적화 대상값 반환을 위한 모델 설정
    model = lgb.LGBMClassifier(**params)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    return accuracy
```

Study 실행

```
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=50)

best_params = study.best_params
best_model = lgb.LGBMClassifier(**best_params)
best_model.fit(X_, y_)
y_pred = best_model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Best Accuracy:", accuracy)
print("Best Parameters:", best_params)
```

Optuna with sklearn

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.svm import SVC

# 목적 함수 구성
def objective(trial):
    # 탐색을 위한 범위 설정
    model_type = trial.suggest_categorical('model_type', ['KNeighbors', 'DecisionTree', 'RandomForest', 'AdaBoost', 'SVC'])

    if model_type == 'KNeighbors':
        n_neighbors = trial.suggest_int('n_neighbors', 3, 10)
        model = KNeighborsClassifier(n_neighbors=n_neighbors)

    elif model_type == 'DecisionTree':
        max_depth = trial.suggest_int('max_depth', 3, 10)
        model = DecisionTreeClassifier(max_depth=max_depth)

    elif model_type == 'RandomForest':
        n_estimators = trial.suggest_int('n_estimators', 100, 1000)
        max_depth = trial.suggest_int('max_depth', 3, 10)
        model = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth)
```

Optuna with sklearn

```
    elif model_type == 'AdaBoost':
        n_estimators = trial.suggest_int('n_estimators', 100, 1000)
        learning_rate =  trial.suggest_float('learning_rate', 0.01, 0.1, log=True)
        # learning_rate = trial.suggest_loguniform('learning_rate', 0.01, 0.1)
        model = AdaBoostClassifier(n_estimators=n_estimators, learning_rate=learning_rate)

    elif model_type == 'SVC':
        C = trial.suggest_loguniform('C', 0.1, 10)
        kernel = trial.suggest_categorical('kernel', ['linear', 'rbf', 'poly'])
        model = SVC(C=C, kernel=kernel)

    else:
        raise ValueError("Invalid model type!")

    # 모델 학습 및 예측
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    accuracy = accuracy_score(y_val, y_pred)
    return accuracy
```

Optuna with sklearn

```
# Optuna를 사용한 하이퍼파라미터 최적화
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=50)

# 최적의 하이퍼파라미터와 정확도 출력
best_params = study.best_params
best_accuracy = study.best_value
print("Best Parameters:", best_params)
print("Best Accuracy:", best_accuracy)
```