```python
import numpy as np
import os
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report
import json

# 전처리한 파일 이용
base_path = r"C:\Users\minar\OneDrive\바탕 화면\학교\기계학습\팀프로젝트\dataset"
X_train_path = os.path.join(base_path, "X_train.npy")
y_train_path = os.path.join(base_path, "y_train.npy")
X_test_path = os.path.join(base_path, "X_test.npy")
y_test_path = os.path.join(base_path, "y_test.npy")

# Numpy 배열 로드
X_train_np = np.load(X_train_path)
y_train_np = np.load(y_train_path)
X_test_np = np.load(X_test_path)
y_test_np = np.load(y_test_path)

# XGBoost 모델 정의
model = XGBClassifier(
    objective='multi:softmax',
    num_class=11, #전체 클래스 갯수 11개
    eval_metric='mlogloss',
    verbosity=1
)
```

1. **데이터 로드**

- 전처리된 학습 데이터와 테스트 데이터를 Numpy 형식으로 불러옵니다.
- 입력 데이터 X_train, X_test와 레이블 y_train, y_test를 각각 로드합니다.

2. **모델 초기화**

- **XGBoost 모델**을 초기화하며, 다중 클래스 분류를 위해 **objective='multi:softmax'**를 설정했습니다.
- **num_class=11**: 클래스 개수가 11개임을 명시하였습니다.
- eval_metric='mlogloss': 다중 클래스 분류에서 손실 함수로 **로그 손실**(Log Loss)을 사용합니다.

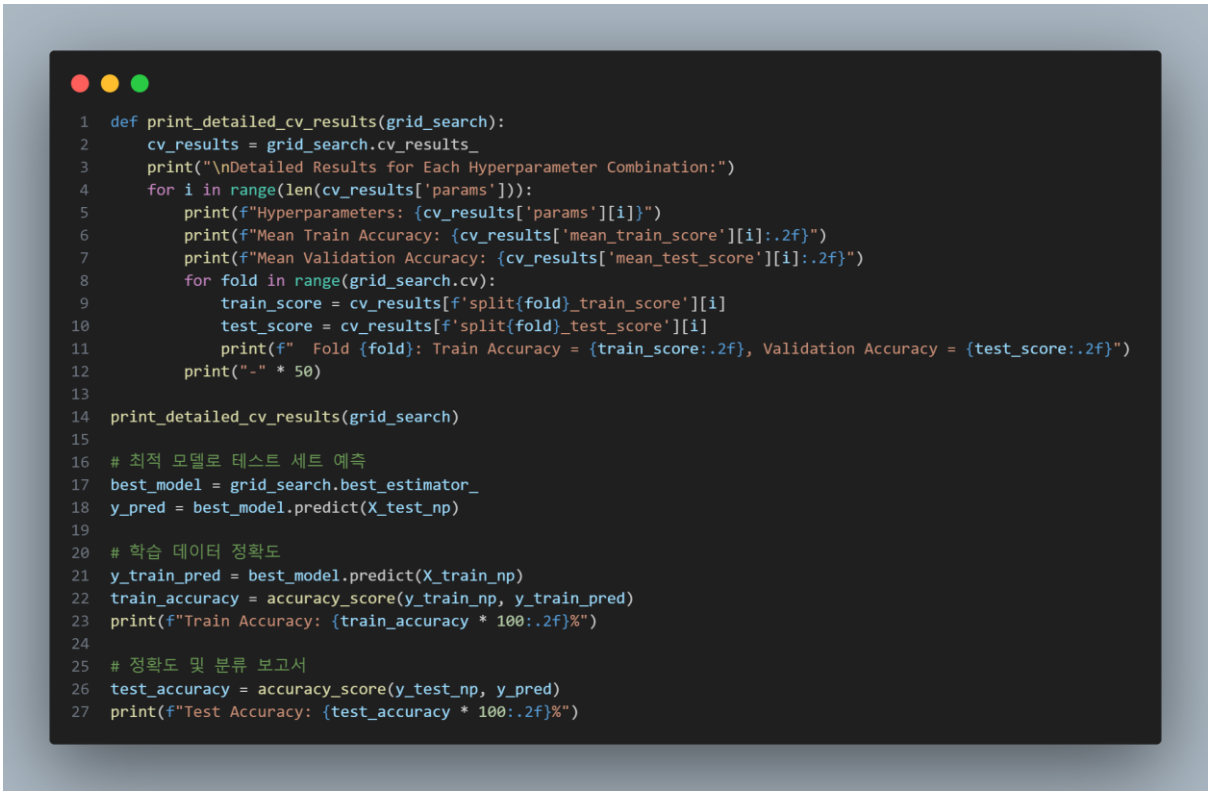
```python
# 하이퍼파라미터 그리드 정의
param_grid = {
    'max_depth': [3, 6],
    'eta': [0.1, 0.05, 0.01],
    'n_estimators': [50, 100, 150],
}

# GridSearchCV 정의
grid_search = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    scoring='accuracy',
    cv=3,
    verbose=1,
    n_jobs=-1,
    return_train_score=True
)

# 그리드 서치 학습
print("GridSearchCV 시작...")
grid_search.fit(X_train_np, y_train_np)

# 최적 하이퍼파라미터 출력
print(f"최적 하이퍼파라미터: {grid_search.best_params_}")
print(f"Best Cross-Validated Accuracy: {grid_search.best_score_:.2f}")

```

1. **하이퍼파라미터 탐색**
   - XGBoost의 주요 하이퍼파라미터를 그리드 탐색(Grid Search)을 통해 최적화합니다.
   - **탐색할 파라미터**:
     - max_depth: 트리의 최대 깊이 (3, 6)
     - eta: 학습률 (0.1, 0.05, 0.01)
     - n_estimators: 트리 개수 (50, 100, 150)
2. **GridSearchCV**
   - 교차 검증을 사용해 모든 파라미터 조합을 평가합니다.
   - **cv=3**: 3-fold 교차 검증을 수행합니다.

- **scoring='accuracy'**: 정확도를 기준으로 최적의 조합을 찾습니다.

· **결과 출력**

- 최적의 하이퍼파라미터와 **검증 정확도(Mean CV Accuracy)**를 출력합니다.



```python
def print_detailed_cv_results(grid_search):
    cv_results = grid_search.cv_results_
    print("\nDetailed Results for Each Hyperparameter Combination:")
    for i in range(len(cv_results['params'])):
        print(f"Hyperparameters: {cv_results['params'][i]}")
        print(f"Mean Train Accuracy: {cv_results['mean_train_score'][i]:.2f}")
        print(f"Mean Validation Accuracy: {cv_results['mean_test_score'][i]:.2f}")
        for fold in range(grid_search.cv):
            train_score = cv_results[f'split{fold}_train_score'][i]
            test_score = cv_results[f'split{fold}_test_score'][i]
            print(f"  Fold {fold}: Train Accuracy = {train_score:.2f}, Validation Accuracy = {test_score:.2f}")
        print("-" * 50)

print_detailed_cv_results(grid_search)

# 최적 모델로 테스트 세트 예측
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_np)

# 학습 데이터 정확도
y_train_pred = best_model.predict(X_train_np)
train_accuracy = accuracy_score(y_train_np, y_train_pred)
print(f"Train Accuracy: {train_accuracy * 100:.2f}%")

# 정확도 및 분류 보고서
test_accuracy = accuracy_score(y_test_np, y_pred)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

1. **세부 결과 출력**

- cv_results_를 이용해 각 하이퍼파라미터 조합의 **Train/Validation Accuracy**를 출력합니다.
- 각 Fold별 교차 검증 결과도 함께 출력하여 모델 성능을 세부적으로 확인합니다.

2. **최적 모델 예측 및 평가**

- **최적의 하이퍼파라미터**를 적용한 모델로 테스트 데이터 예측을 수행합니다.
- 학습 정확도(Train Accuracy)와 테스트 정확도(Test Accuracy)를 계산하여 출력합니다.

```python
save_path = os.path.join(base_path, "xgboost_best_model_.json")
best_model.save_model(save_path)
print(f"Best model saved to {save_path}")

# 메타데이터 생성, 실험 결과에 대한 간략한 데이터
cv_results = grid_search.cv_results_
model_metadata = {
    "test_accuracy": test_accuracy,
    "train_accuracy": train_accuracy,
    "classification_report": classification_report(y_test_np, y_pred, output_dict=True),
    "best_hyperparameters": grid_search.best_params_,
    "best_cv_accuracy": grid_search.best_score_,
    "cv_results": [
        {
            "params": cv_results['params'][i],
            "mean_train_score": cv_results['mean_train_score'][i],
            "mean_test_score": cv_results['mean_test_score'][i],
            "fold_scores": {
                f"fold_{fold}": {
                    "train": cv_results[f'split{fold}_train_score'][i],
                    "validation": cv_results[f'split{fold}_test_score'][i]
                }
                for fold in range(grid_search.cv)
            }
        }
        for i in range(len(cv_results['params']))
    ]
}

# 메타데이터 저장
metadata_save_path = os.path.join(base_path, "xgboost_best_model_metadata.json")
with open(metadata_save_path, "w") as f:
    json.dump(model_metadata, f, indent=4)
print(f"Metadata saved to {metadata_save_path}")

```

1. **모델 저장**
   - 최적의 하이퍼파라미터로 학습된 모델을 **JSON 형식**으로 저장합니다.
   - 저장된 모델은 나중에 불러와 바로 예측에 사용할 수 있습니다.
2. **메타데이터 저장**
   - 실험 결과를 기록한 **메타데이터 파일**을 JSON 형식으로 저장합니다.
   - 기록된 내용:
     - 테스트 및 학습 정확도
     - 최적 하이퍼파라미터
     - 교차 검증 결과

- 분류 보고서

3. 결과 확인

- 저장된 모델 및 메타데이터 파일은 다른 실험과 비교하거나 후속 분석에 활용할 수 있습니다.