**Optuna**

(<https://optuna.readthedocs.io/en/stable/>)

머신러닝 모델의 하이퍼파라미터를 자동으로 탐색하는 파이썬 기반 최적화 라이브러리

**- 필요성**

머신러닝 모델 성능은 하이퍼파라미터 조합에 크게 영향을 받음

max\_depth, n\_estimators, learning\_rate 등 하이퍼파라미터 값 하나하나가 결과에 영향을 줌

따라서 최적의 결과를 위한 조합 가능한 경우의 수는 하이퍼파라미터 개수, 값의 범위가 늘어날수록 기하급수적으로 증가 → 수동으로는 최적의 하이퍼파라미터 조합 사실상 불가능

모델 선택, 가중치, 하이퍼파라미터 수동 조정은 매우 번거롭고 비효율적이며 조정하는 사람에 따라 모델 성능의 차이가 큼

-> Optuna는 모델 조합, 가중치, 하이퍼파라미터 값이 적합한지 자동으로 탐색하므로

높은 인적 편차와 수작업 의존도 해결 가능

또한, Optuna는 교차검증 기반 평가(K-fold cross validation)를 통해 과적합을 방지하며 일반화 성능이 가장 좋은 조합을 선택함 (optuna\_vc\_optimization.py 에서는 3-Fold Stratified Cross-Validation 사용)

**- 내부 구조**

**1. 경고제거**

불필요한 경고 메시지 제거

\* warnings.simplefilter("ignore"): Python 경고 무시

\* os.environ["PYTHONWARNINGS"] = "ignore": 환경 변수 차원에서도 무시

\* optuna.logging.set\_verbosity(optuna.logging.WARNING): Optuna의 로그를 warning 수준 이하로 제한

**2. 경로 설정**

데이터, 결과 저장 경로 관리 일원화

\* BASE\_DIR: 현재 실행 중인 스크립트의 경로

\* DATA\_PATH: 입력 데이터셋 (2025\_URL\_DATASET.csv)

\* BEST\_PARAMS\_PATH: 최종 결과 저장 위치 (best\_params.json)

\* CHECKPOINT\_PATH: 중간 결과 저장 위치 (optuna\_checkpoint.json)

**3. 데이터셋 로드**

데이터셋을 불러와 입력(X), 정답(y) 분리

**4. 전처리**

누락값 보정 및 정규화

\* SimpleImputer(strategy="mean"): 결측값을 평균으로 대체

\* StandardScaler(): 평균 0, 표준편차 1로 정규화

\* Pipeline으로 묶어서 적용

데이터셋을 학습용/테스트용으로 나누되 클래스 비율 유지

\* test\_size=0.2: 80% 학습, 20% 테스트

\* StratifiedShuffleSplit: 레이블 비율이 유지되도록 섞어서 분리

**6. trial 내부 fold 학습/검증 출력**

한 trial 모델의 성능을 3-Fold Stratified Cross Validation으로 평가

\* 각 fold마다 fit → predict → f1\_macro 계산

\* 결과를 평균하여 하나의 점수 반환

→ 과적합 방지, 안정적인 성능 평가

**7. Optuna 목적 함수 정의**

한 번의 trial에 대해

\* 14개의 모델 중 어떤 모델들을 조합할지 결정

\* 선택된 모델마다 하이퍼파라미터 + 가중치(weight)도 같이 선택

\* soft voting 시에는 weights 적용

\* 모델 학습 후 cross-validation으로 성능 평가하여 점수 반환

→ Optuna가 자동으로 이 함수 수천 번 반복하여 최적 조합 탐색

**8. 탐색 횟수**

Optuna 최적화 반복 횟수 설정

\* n\_trials = 2000 설정 (반복 횟수 임의 변경 가능)

\* study.ask() → trial 구성

\* study.tell() → 평가 점수 기록

\* 반복 도중 중간 저장(checkpoint) 도 수행

\* 모든 trial 종료 후 checkpoint 삭제

**9. 결과 출력**

\* study.best\_trial을 통해 가장 성능 좋은 조합을 출력

\* 어떤 모델이 선택됐고, 각 모델의 파라미터와 가중치가 어떻게 되었는지 정리

\* best\_params.json에 저장하여 추후 재사용 가능

**- 프로그램 흐름**

CSV 데이터 로드 및 전처리

↓

Train/Test 분할

↓

Optuna Trial 1 ~ N 실행

- 모델 구성

- Voting 방식, weight 설정

- 파라미터 자동 탐색

- 3-Fold 교차검증 평가

↓

Best 조합 선택 및 저장 (json)

**VotingClassifier**

(<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html>)

여러 머신러닝 모델을 조합해서 투표 방식으로 최종 예측을 만드는 앙상블 모델

**- 필요성**

모델마다 단점, 강점이 다름

ex) LogisticRegression은 단순하고 빠름, XGBoost는 비선형 문제에 강함

한 가지 모델만 사용할 경우 특정 유형의 데이터에 대해 오차와 과적합 위험이 크게 증가할 수 있음

서로 다른 특성을 가진 모델들을 조합하면 어떤 데이터든 적어도 하나의 모델이 잘 예측할 가능성이 높아짐 -> 결과적으로 오차를 줄이고 일반화, 일관성을 높이는 효과

**\* Voting 방식으로는 Hard Voting / Soft Voting 존재**

Hard Voting: 각 모델이 예측한 클래스 중 가장 많이 선택된 클래스를 최종 예측값으로 결정

Soft Voting: 각 모델이 예측한 클래스별 확률을 평균하여 가장 높은 확률을 가진 클래스를 선택

Soft voting 방식에서는 각 모델마다 가중치를 부여가능하기에 단순 평균이 아닌 성능, 신뢰도 높은 모델의 예측 결과가 전체 판단에 더 큰 영향력을 갖도록 조절할 수 있음

ex) XGBoost (정확도 95%) → weight 4 / DecisionTree (정확도 70%) → weight 1

→ 예측 결과에 XGBoost의 의견이 더 크게 반영됨

**- 내부 구조**

**1. 경고 제거**

불필요한 경고 메시지 제거

\* warnings.simplefilter("ignore"): 일반 경고 무시

\* os.environ["PYTHONWARNINGS"] = "ignore": 환경변수 경고 무시

**2. 경로 설정**

데이터, 결과 저장 경로 관리 일원화

\* DATA\_PATH: 학습 데이터셋 (2025\_URL\_DATASET.csv)

\* PARAMS\_PATH: Optuna가 저장한 최적 조합 (best\_params.json)

\* MODEL\_PATH: 학습된 최종 모델 저장 위치 (mvc\_model.pkl)

\* PREPROCESSOR\_PATH: 전처리기 저장 위치 (preprocessor.pkl)

**3. 최적 조합 로드**

best\_params.json 에서 모델 종류, 하이퍼파라미터, 가중치, 투표 방식 등을 불러옴

→ Optuna가 자동으로 탐색한 최적의 조합

**4. 데이터셋 로드**

데이터셋을 불러와 입력(X), 정답(y) 분리

**5. 전처리**

누락값 보정 및 정규화

\* SimpleImputer(strategy="mean"): 결측값을 평균으로 대체

\* StandardScaler(): 평균 0, 표준편차 1로 정규화

\* Pipeline으로 묶어서 일괄 적용

데이터셋을 학습용/테스트용으로 나누되 클래스 비율 유지

\* test\_size=0.2: 80% 학습, 20% 테스트

\* StratifiedShuffleSplit: 레이블 비율이 유지되도록 무작위 분리

**6. 모델 구성 함수 정의**

\* 14개의 지원 모델에 대해 함수 get\_model(name, params)로 객체 생성

\* 각 모델은 best\_params에서 해당 파라미터를 가져와 구성됨

**7. VotingClassifier 조합 생성**

\* best\_params 기준으로 어떤 모델들을 사용할지 판단 (use\_xgb, use\_dt 등)

\* 선택된 모델들을 리스트에 추가하며, soft voting인 경우 각각의 weight도 적용

\* 최종적으로 VotingClassifier 객체를 구성

**8. 학습 및 평가**

\* VotingClassifier.fit() 으로 학습 수행

\* 테스트셋에 대해 예측 및 평가 (classification\_report, accuracy\_score)

→ 성능 출력

**9. 모델 및 전처리기 저장**

\* 학습이 완료된 모델을 .pkl 파일로 저장 (joblib.dump)

\* 추후 예측 시스템에서 로드하여 사용 가능

**- 프로그램 흐름**

best\_params.json 로드

↓

CSV 데이터셋 불러오기 및 전처리

↓

Stratified train/test 분할

↓

best\_params에 따라 모델 구성

↓

VotingClassifier 조합 구성 (soft/hard + weight)

↓

학습 → 테스트 → 평가 (정확도 및 분류 리포트)

↓

모델 및 전처리기 저장