치매 EEG 데이터 분류 결과

11.29

임도현

Data description

AD

46-1 subjects,
2 dataset

1850 segments

MCI

11 subjects,
1 dataset

1383 segments

NC

171-1 subjects,
6 dataset

5554 segments

NC 클래스 편향으로 인한 overfitting 우려

→ NC 데이터 절반 소거

Preprocessing

Feature extraction

Kurtosis(첨도)

Band Power (alpha, beta, delta, theta, gamma)

Spectral entropy

각 채널에 대해 = **19*7 = 133 features** → L1 Regularization으로 feature selection

Kurtosis_Fp1 | Kurtosis_Fp2 | Kurtosis_F7 | ... | alpha_Fp1 | ... | Spectral entropy_O2

이러한 방식으로 채널별 피처를 나열

kurtosis_Fp1	kurtosis_Fp2	kurtosis_F7	kurtosis_F3	kurtosis_Fz	kurtosis_F4	kurtosis_F8	kurtosis_T7	kurtosis_C3	kurtosis_Cz	kurtosis_C4	kurtosis_T8	kurtosis_P7	kurtosis_P3	kurtosis_Pz	kurtosis_P4	kurtosis_P8	kurto
-0.4183582	0.33246772	-0.3426999	-0.4347618	-0.3630832	1.54147344	3.80385867	-0.366574	-0.3610341	-0.3170229	-0.3460871	-0.3953567	-0.2962779	-0.3430376	-0.2912622	-0.3173509	-0.3891812	-0.2
-0.5539365	0.64543697	-0.4995246	-0.4132235	-0.2333297	1.56661142	3.6919516	-0.4466041	-0.1958969	-0.0959387	-0.204263	-0.4872127	-0.4988442	-0.3265615	-0.2288259	-0.2625676	-0.4930295	-0.4
1.02831965	-0.381386	0.39092526	0.61200935	0.00739263	-1.0330334	-0.2639297	-0.2696172	-0.0036998	-1.8066105	-0.9358068	-0.9268778	0.70892068	0.9442505	0.00990863	0.69891827	0.38099317	-1.6
1.02471203	-0.3508402	0.4694742	0.55502302	-0.0596957	-0.8648803	0.11284285	-0.260772	-0.0607672	-1.8786024	-1.0169827	-0.9903204	0.6907373	0.91469457	-0.0201844	0.6367612	0.33877663	-1.7

Preprocessing

- Data

(Data) Version 1

: NC 데이터에서 0~85번까지 사용,
나머지는 test set II로 활용

(Data) Version 2

: Mixed NC Dataset에 섞인 여러 개별 데이터셋의
각각 절반씩 모아 사용, 나머지는 test set II로 활용

1~16 (16개)	(1~8)
17~65 (49개)	(17~41)
66~116 (51개)	(66~91)
117~131 (15개)	(117~124)
132~142 (11개)	(132~137)
143~171 (29개)	(143~158)

- Model

(Model) Version 1

Multi-class classification으로 접근
하나의 모델이 NC/MCI/AD 중 하나로 분류
Random Forest(RF), eXtreme Gradient Boost(XGB) 사용

(Model) Version 2

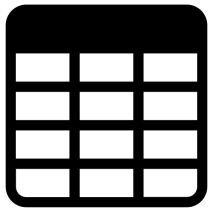
Binary classification으로 접근
하나의 모델이 NC/MCI 또는 NC/AD를 분류
Random Forest(RF), eXtreme Gradient Boost(XGB) 사용

(Model) Version 3

Binary classification으로 접근
하나의 모델이 NC/MCI 또는 NC/AD를 분류
Common Spatial Pattern 사용
Random Forest(RF), eXtreme Gradient Boost(XGB) 사용

<세그먼트 수가 많지 않기에 DL Method는 overfitting
예상되어 사용하지 않음>

Training



rows(세그먼트 개수)
*
cols(피처 개수)
(전처리된 데이터)

Shuffle
(한 클래스 데이터
특정 위치에
몰리지 않도록)

Train-Test split
0.8 / 0.2

L1 Regularization

모델 정의
(Random Forest /
XGB)

Grid Search
파라미터 정의

```
param_grid = {
    'feature_selection__estimator__alpha': [0.001, 0.01, 0.1, 1],  # Lasso의 alpha 값
    'random_forest__n_estimators': [50, 100, 200],  # Random Forest의 트리 개수
    'random_forest__max_depth': [None, 10, 20, 30]  # Random Forest의 최대 깊이
}
```

```
param_grid = {
    'feature_selection__estimator__alpha': [0.001, 0.01, 0.1],
    'xgboost__n_estimators': [50, 100, 200],
    'xgboost__max_depth': [3, 5, 7],
    'xgboost__learning_rate': [0.01, 0.1, 0.3]
}
```

fit()

Test set으로
test

나머지 분리되었던
NC dataset으로
test

Feature extraction

extract_features.py

```python
def extract_features(file_path, label, channels, bands, normalize_kurtosis=True):
    """
    단일 EDF 파일에서 특징 추출 후 데이터프레임으로 반환
    """
    # EDF 파일 로드
    raw = mne.io.read_raw_edf(file_path, preload=True)
    if channels:
        raw.pick_channels(channels)

    data = raw.get_data()  # (n_channels, n_samples)
    sfreq = raw.info['sfreq']

    features = {}
    # 시간 영역 특징
    kurtosis_values = [kurtosis(data[ch_idx]) for ch_idx in range(data.shape[0])]
    if normalize_kurtosis:
        scaler = StandardScaler()  # Min-Max 스케일링 (0~1)
        kurtosis_values = scaler.fit_transform(np.array(kurtosis_values).reshape(-1, 1)).flatten()
    for ch_idx, ch_name in enumerate(raw.ch_names):
        features[f"kurtosis_{ch_name}"] = kurtosis_values[ch_idx]
    # 주파수 대역 특징
    band_powers = compute_band_power(data, sfreq, bands)
    for band_name, power_values in band_powers.items():
        for ch_idx, ch_name in enumerate(raw.ch_names):
            features[f"{band_name}_{ch_name}"] = power_values[ch_idx]

    # Spectral Entropy
    for ch_idx, ch_name in enumerate(raw.ch_names):
        features[f"spectral_entropy_{ch_name}"] = spectral_entropy(data[ch_idx], sfreq, method='welch')

    # DataFrame 생성
    feature_df = pd.DataFrame([features])

    # label 추가 및 마지막 열로 이동
    feature_df['label'] = label
    feature_df = feature_df[[col for col in feature_df.columns if col != 'label'] + ['label']]

    return feature_df
```

```python
def compute_band_power(data, sfreq, bands):
    """
    EEG 데이터의 주파수 대역별 Band Power를 계산합니다.

    Args:
        data (np.ndarray): (n_channels, n_samples) 형태의 EEG 데이터.
        sfreq (float): 샘플링 주파수 (Hz).
        bands (dict): 주파수 대역 이름과 범위를 매핑한 딕셔너리.

    Returns:
        dict: 주파수 대역별 Band Power.
    """
    band_powers = {}
    freqs, psd = welch(data, fs=sfreq, nperseg=sfreq*2, axis=1)
    for band_name, (fmin, fmax) in bands.items():
        band_idx = np.logical_and(freqs >= fmin, freqs <= fmax)
        band_power = np.sum(psd[:, band_idx], axis=1)  # 각 채널별로 계산
        band_powers[band_name] = band_power
    return band_powers
```

```python
def process_group(group_path, label, channels, bands):
    """
    그룹(AD, MCI, NC)의 모든 EDF 파일에 대해 특징 추출
    """
    feature_frames = []
    for subfolder in os.listdir(group_path):
        subfolder_path = os.path.join(group_path, subfolder)
        if os.path.isdir(subfolder_path):
            for file in os.listdir(subfolder_path):
                if file.endswith(".edf"):
                    edf_path = os.path.join(subfolder_path, file)
                    print(f"Processing {edf_path}")
                    feature_df = extract_features(edf_path, label, channels, bands)
                    feature_frames.append(feature_df)
    return pd.concat(feature_frames, ignore_index=True)
```

1) AD 폴더 추출 후 Dataframe 생성

2) MCI 폴더 추출 후 Dataframe 생성

...

3) 추출된 Dataframe 병합

Feature extraction

extract_features_csp.py

```python
for group_name, label in groups.items():
    group_path = os.path.join(base_dir, group_name)
    group_features = process_group(group_path, label, channels, bands)
    all_features.append(group_features)
# 모든 그룹 데이터를 하나로 병합
final_df = pd.concat(all_features, ignore_index=True)

# final_df에서 label열을 따로 분리
labels = final_df.pop('label')

# Common Spatial Pattern
csp = extract_csp(edf_dir, csp_label_dir, channels)
# csp값을 feature에 추가
for i in range(csp.shape[1]):
    final_df[f'csp{i}'] = csp[:, i]

# 분리한 label열을 다시 추가
final_df['label'] = labels
final_df.to_csv(output_csv, index=False)
```

```python
def extract_csp(data_dir, label, channels):
    labels = np.load(label)
    chunk_size= 30*250

    y = transform_labels_to_trials(labels, chunk_size=30*250)
    print(f"Labels shape: {y.shape}")

    # MNE Epochs 객체로 변환
    epochs, _ = edf_to_epochs(data_dir, channels=channels, chunk_size=chunk_size, labels=y)
    X = epochs.get_data()  # (n_trials, n_channels, n_times)
    csp = CSP(n_components=4, reg=None, log=True, norm_trace=False)
    X_new = csp.fit_transform(X, y)
    return X_new
```

```python
def extract_features_binary(file_path, label, channels, normalize_kurtosis=True):
    """
    단일 EDF 파일에서 특징 추출 후 데이터프레임으로 반환
    """
    # EDF 파일 로드
    raw = mne.io.read_raw_edf(file_path, preload=True)
    if channels:
        raw.pick_channels(channels)

    data = raw.get_data()  # (n_channels, n_samples)
    sfreq = raw.info['sfreq']

    features = {}
    # 시간 영역 특징
    kurtosis_values = [kurtosis(data[ch_idx]) for ch_idx in range(data.shape[0])]
    if normalize_kurtosis:
        scaler = StandardScaler()  # Min-Max 스케일링 (0~1)
        kurtosis_values = scaler.fit_transform(np.array(kurtosis_values).reshape(-1, 1)).flatten()
    for ch_idx, ch_name in enumerate(raw.ch_names):
        features[f"kurtosis_{ch_name}"] = kurtosis_values[ch_idx]

    """
    # 주파수 대역 특징
    band_powers = compute_band_power(data, sfreq, bands)
    for band_name, power_values in band_powers.items():
        for ch_idx, ch_name in enumerate(raw.ch_names):
            features[f"{band_name}_{ch_name}"] = power_values[ch_idx]
    """
# Spectral Entropy
for ch_idx, ch_name in enumerate(raw.ch_names):
    features[f"spectral_entropy_{ch_name}"] = spectral_entropy(data[ch_idx], sfreq, method='welch')

# DataFrame 생성
feature_df = pd.DataFrame([features])

# label 추가 및 마지막 열로 이동
feature_df['label'] = label
feature_df = feature_df[[col for col in feature_df.columns if col != 'label'] + ['label']]

return feature_df
```

Model

Random Forest

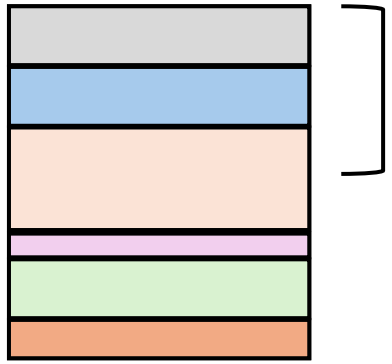
```
# Lasso를 사용한 특징 선택
lasso = Lasso(max_iter=10000)

# Random Forest 모델
rf_model = RandomForestClassifier(random_state=42)

# 파이프라인 구축
pipeline = Pipeline([
    ('feature_selection', SelectFromModel(estimator=lasso)),
    ('random_forest', rf_model)
])

# 하이퍼파라미터 그리드 정의
param_grid = {
    'feature_selection__estimator__alpha': [0.001, 0.01, 0.1, 1],  # Lasso의 alpha 값
    'random_forest__n_estimators': [50, 100, 200],  # Random Forest의 트리 개수
    'random_forest__max_depth': [None, 10, 20, 30]  # Random Forest의 최대 깊이
}
# Weighted F1 스코어 사용
f1_scorer = make_scorer(f1_score, average='weighted')

grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring=f1_scorer, n_jobs=-1, verbose=1)
```

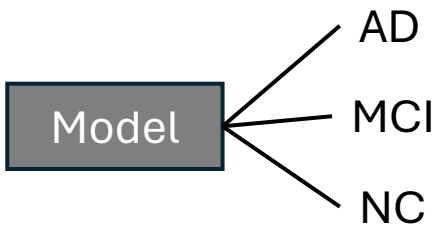
동일한 구조로 모든 버전 학습

XGB

```
# Lasso를 사용한 특징 선택
lasso = Lasso(max_iter=10000)

# XGBoost 모델
xgb_model = XGBClassifier(
    objective='binary:logistic',
    use_label_encoder=False,
    eval_metric='logloss'
)

# 파이프라인
pipeline = Pipeline([
    ('feature_selection', SelectFromModel(estimator=lasso)),
    ('xgboost', xgb_model)
])

# Grid Search 설정
param_grid = {
    'feature_selection__estimator__alpha': [0.001, 0.01, 0.1],
    'xgboost__n_estimators': [50, 100, 200],
    'xgboost__max_depth': [3, 5, 7],
    'xgboost__learning_rate': [0.01, 0.1, 0.3]
}

scoring = {
    'accuracy': make_scorer(accuracy_score),
    'f1_score': make_scorer(f1_score, average='weighted')
}
grid_search_xgb = GridSearchCV(
    pipeline,
    param_grid,
    cv=5,
    scoring=scoring,
    refit='accuracy',
    verbose=1,
    n_jobs=-1
)
```

Training





Version 1 (안 섞인 NC + Multi-class)

- Random Forest (L1 + Grid)

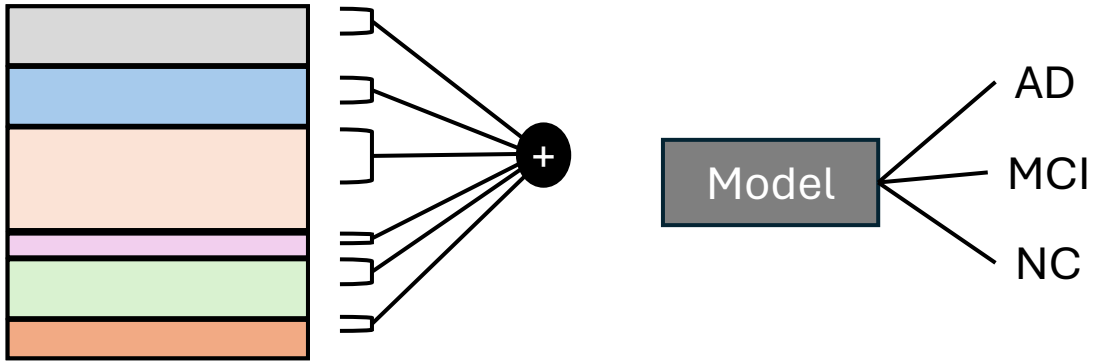
```
Best Parameters: {'feature_selection__estimator__alpha': 0.001, 'random_forest__max_depth': None,
 'random_forest__n_estimators': 200}
Best CV Score: 0.9359792167476293
Model saved to 'best_pipeline_model.pkl'
Test Accuracy: 0.94 Test F1: 0.94
```

- XGB (L1 + Grid)

```
Best Parameters: {'feature_selection__estimator__alpha': 0.001, 'xgboost__learning_rate': 0.3, 'xgboost__max_depth': 5,
 'xgboost__n_estimators': 200}
Test Accuracy: 0.96
Test F1-Score: 0.96
```

두 모델 다 특정 데이터셋에 편향되어 나머지 NC 데이터에 대해 F1-score 0.46 정도 값 기록함
-> 데이터셋 섞어서 다시 시도

Training



Version 2 (섞인 NC + Multi-class)

< AD vs NC >

- Random Forest (L1 + Grid)

```
Best Parameters: {'feature_selection__estimator__alpha': 0.001, 'random_forest__max_depth': 30, 'random_forest__n_estimators':
  200}
Best CV Score: 0.8665789224704656
Model saved to 'best_l1rf_model_ver2.pkl'
Test Accuracy: 0.88 Test F1: 0.88
```

- XGB (L1 + Grid)

```
Best Parameters: {'feature_selection__estimator__alpha': 0.001, 'xgboost__learning_rate': 0.3, 'xgboost__max_depth': 7,
 'xgboost__n_estimators': 200}
Test Accuracy: 0.89
Test F1-Score: 0.89
```

나머지 NC 데이터로 시도했을 때

```
rf. Test Accuracy: 0.60 Test F1: 0.75
xgb. Test Accuracy: 0.62 Test F1: 0.76
```

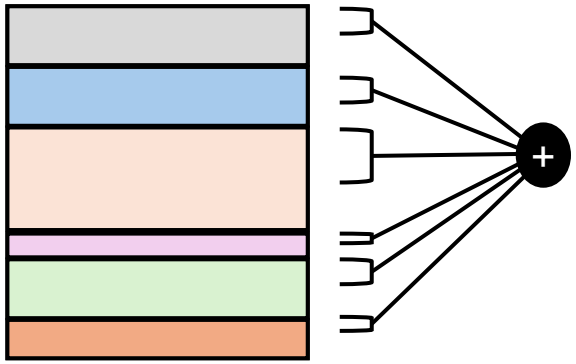
AD/CN	F1
LightGBM	76.12%
SVM	73.74%
kNN	72.81%
MLP	74.82%
Random Forests	75.31%

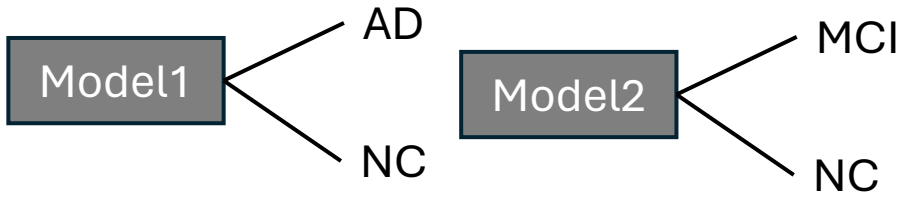
AD 1번 Dataset 제공 논문

(**A Dataset of Scalp EEG Recordings of Alzheimer's Disease, Frontotemporal Dementia and Healthy Subjects from Routine EEG**)

과 비슷한 성능 보여줌

Training





Version 3 (섞인 NC + Binary-class)

< AD vs NC >

- Random Forest (L1 + Grid)

```
Best Parameters: {'feature_selection__estimator__alpha': 0.001, 'random_forest__max_depth': 30,
 'random_forest__n_estimators': 200}
Best CV Score: 0.8853719597013823
Model saved to 'best_l1rf_model_ver3.pkl'
Test Accuracy: 0.91 Test F1: 0.91
```

- XGB (L1 + Grid)

```
Best Parameters: {'feature_selection__estimator__alpha': 0.001, 'xgboost__learning_rate': 0.3, 'xgboost__max_depth': 7,
 'xgboost__n_estimators': 200}
Test Accuracy: 0.92
Test F1-Score: 0.92
```

< NC2 Dataset >

```
rf. Test Accuracy: 0.84 Test F1: 0.91
xgb. Test Accuracy: 0.83 Test F1: 0.90
```

< MCI vs NC >

- Random Forest (L1 + Grid)

```
Best Parameters: {'feature_selection__estimator__alpha': 0.001, 'random_forest__max_depth': None, 'random_forest__n_estimators':
 200}
Best CV Score: 0.9351924021634158
Model saved to 'best_l1rf_model_ver3_mci.pkl'
Test Accuracy: 0.94 Test F1: 0.94
```

- XGB (L1 + Grid)

```
Best Parameters: {'feature_selection__estimator__alpha': 0.001, 'xgboost__learning_rate': 0.3, 'xgboost__max_depth': 5,
 'xgboost__n_estimators': 200}
Test Accuracy: 0.94
Test F1-Score: 0.94
```

< NC2 Dataset >

```
rf. Test Accuracy: 0.74 Test F1: 0.85
xgb. Test Accuracy: 0.78 Test F1: 0.87
```

Training

L1 Regularization

```
array([False,  True,  True,  True, False,  True,  True,  True, False,
       False, False,  True,  True, False, False,  True,  True, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False,  True,  True,
        True,  True, False, False, False, False, False,  True,  True,
        True,  True, False, False, False,  True,  True])
```

Kurtosis 일부, Spectral entropy 일부만 사용되고 있음을 알 수 있음

band power는 version 4에서 모두 소거

Training

Version 4 (섞인 NC + Binary-class + CSP)

두 클래스 간 공간 분산 차이를 최대화
→ 한 클래스 분산은 키우고, 다른 클래스 분산은 줄임

→ 특정 한 클래스가 잘 구분되도록 만들 수 있음

그러나 이진 클래스에 대해서만 수행 가능
(= NC, MCI, AD를 모두 한 번에 구별하도록 반들 수 없음)

MNE Install Documentation API Reference Get help Development Choose version

Section Navigation
- Most-used classes
- Reading raw data
- File I/O
- Creating data objects from arrays
- Exporting
- Datasets
- Visualization

mne.decoding.CSP

```
class mne.decoding.CSP(n_components=4, reg=None, log=None, cov_est='concat',
transform_into='average_power', norm_trace=False, cov_method_params=None,
rank=None, component_order='mutual_info')
```
[source]

M/EEG signal decomposition using the Common Spatial Patterns (CSP). This class can be used as a supervised decomposition to estimate spatial filters for feature extraction. CSP in the context of EEG was first described in [1]; a comprehensive tutorial on CSP can be found in [2]. Multi-class solving is implemented from [3].

Parameters:

On this page
CSP
Examples using
mne.decoding.CSP

mne 라이브러리 내장 기능 사용

`fit_transform(X, y, **fit_params)` [source]

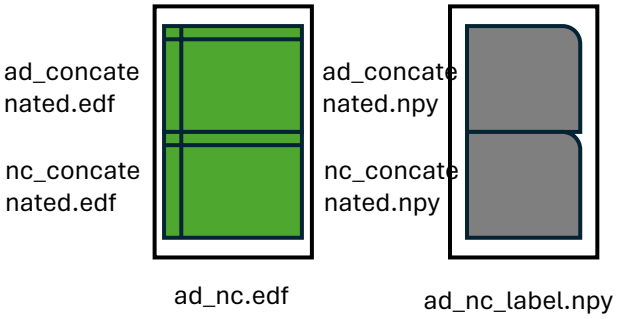
Fit to data, then transform it. Fits transformer to `X` and `y` with optional parameters `fit_params`, and returns a transformed version of `X`.

Parameters:

X : *array*, *shape (n_samples, n_features)*
Training set.

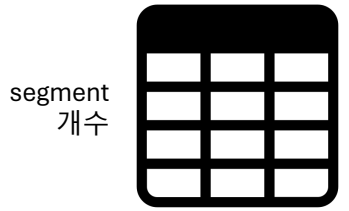
y : *array*, *shape (n_samples,)*
Target values or class labels.

****fit_params** : *dict*
Additional fitting parameters passed to the `fit` method..



CSP.fit_transform()

4(csp component 개수)



기존 피처 Dataframe에 병합

Training

Version 4 (섞인 NC + Binary-class + CSP)

< AD vs NC >

- Random Forest (L1 + Grid)

```
Best Parameters: {'feature_selection__estimator__alpha': 0.001, 'random_forest__max_depth': 20, 'random_forest__n_estimators': 100}
Best CV Score: 0.8918876142473652
Model saved to 'best_l1rf_model_ver4.pkl'
Test Accuracy: 0.89 Test F1: 0.89
```

- XGB (L1 + Grid)

```
Best Parameters: {'feature_selection__estimator__alpha': 0.001, 'xgboost__learning_rate': 0.3, 'xgboost__max_depth': 7,
 'xgboost__n_estimators': 200}
Test Accuracy: 0.92
Test F1-Score: 0.92
```

< AD vs MCI >

- Random Forest (L1 + Grid)

```
Best Parameters: {'feature_selection__estimator__alpha': 0.001, 'random_forest__max_depth': None, 'random_forest__n_estimators':
 200}
Best CV Score: 0.8908448382674281
Model saved to 'best_l1rf_model_ver4_csp_mci.pkl'
Test Accuracy: 0.89 Test F1: 0.89
```

- XGB (L1 + Grid)

```
Best Parameters: {'feature_selection__estimator__alpha': 0.001, 'xgboost__learning_rate': 0.3, 'xgboost__max_depth': 7,
 'xgboost__n_estimators': 200}
Test Accuracy: 0.91
Test F1-Score: 0.91
```

CSP는 두 개 이상의 클래스를 가지는 데이터셋에 대해서 수행 가능한 기법

(NC2만 따로 전처리 불가)

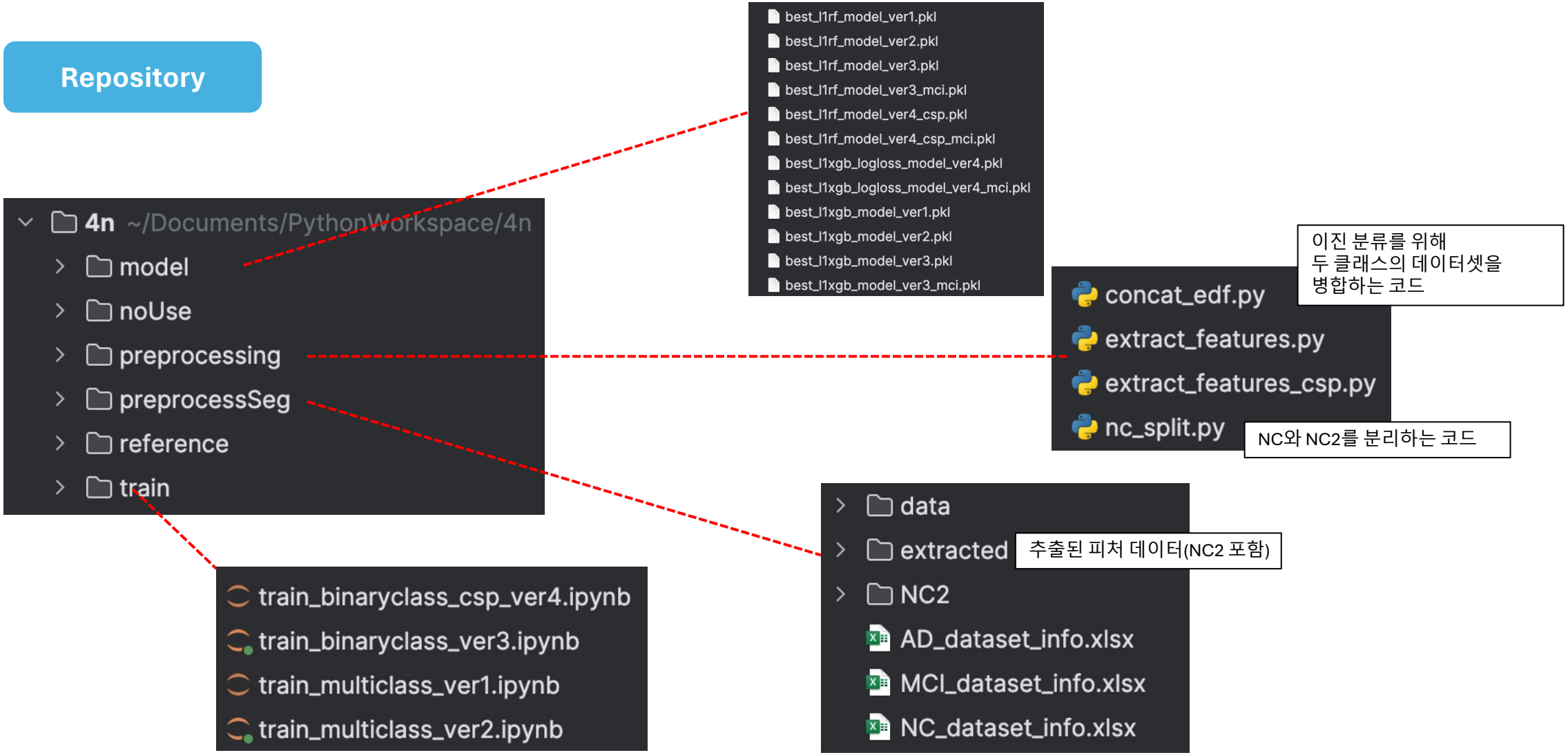
Summary

	Version 1 (Multi-class)		Version 2 (Multi-class)		Version 3				Version 4			
					(AD vs NC)		(MCI vs NC)		(AD vs NC)		(MCI vs NC)	
Model	**RF**	**XGB**	**RF**	**XGB**	**RF**	**XGB**	**RF**	**XGB**	**RF**	**XGB**	**RF**	**XGB**
Accu-racy	0.94	0.96	0.88	0.89	0.89	0.92	0.94	0.94	0.89	0.92	0.89	0.91
F1	**0.94**	**0.96**	**0.88**	**0.89**	**0.89**	**0.92**	**0.94**	**0.94**	**0.89**	**0.92**	**0.89**	**0.91**
(NC2) Accu-racy	0.30	0.28	0.60	0.62	0.84	0.83	0.74	0.78	-	-	-	-
(NC2) F1	**0.46**	**0.44**	**0.75**	**0.76**	**0.91**	**0.90**	**0.85**	**0.87**	-	-	-	-

- CSP만을 피처로 사용하는 케이스, CSP를 다른 피처와 함께 사용하는 케이스 모두 성능이 개선되지 않음

- (다른 성능 개선 방안 - Focal Loss)
    - 클래스 불균형 데이터를 완화해 주는 Loss function, 예측하기 쉬운 클래스에 더 적은 loss를 줌
    - 본 프로젝트 데이터의 경우 NC 클래스로 편향되어 있어, 성능 개선 가능성 있어 보임

Repository



ver 1, 2, 3: extract_features로 피처 데이터 추출(preprocessing.extracted 에 저장됨) → train 폴더의 주피터로 train

ver 4: CSP를 위한 edf 및 npy(라벨) 추출 → extract_features_csp)로 피처 데이터 추출(preprocessing.extracted 에 저장됨)
→ train 폴더의 주피터로 train