Acute Dermal
Machine Learning Code 설명

20210093 김가언

Train code-데이터 로드

```python
def load_sdf_to_df(filename):
    suppl = Chem.SDMolSupplier(filename)
    rows = []
    for mol in suppl:
        if mol is not None:
            row = {prop: mol.GetProp(prop) for prop in mol.GetPropNames()}
            row['SMILES'] = Chem.MolToSmiles(mol)
            rows.append(row)
    return pd.DataFrame(rows)
```

Chem.SDMolSupplier
: Rdkit 라이브러리의 클래스,
SDF파일에서 화학분자를 순차적으로 읽음

mol.GetPropNames()
: 해당 화학 분자에 저장된 모든 속성의
이름을 가져옴

mol.GetProp(prop)
: 각 속성의 값을 가져옴

Chem.MolToSmiles:
화학분자를 smiles로 변환

```python
file="/content/drive/MyDrive/train_set_acute_dermal_features_rdkitcdk.sdf"
train_df = load_sdf_to_df(file)
```

Train SDF 파일에 저장된 화학 분자의 속성과 SMILES 표기가 포함된 데이터프레임으로 변환하여 trian_df에 저장

Train code-데이터 전처리

```python
def string_to_list(bit_string):
    if isinstance(bit_string, str):
        return list(map(int, bit_string.strip('[]').split(', ')))
    else:
        return bit_string
```

isinstance(bit_string, str)
: 입력값이 문자열인지 확인
참이면, 문자열에서 대괄호 제거하고 콤마를 기준으로 분할
거짓이면, 입력값을 그대로 반환

```python
train_df['MACCS_Descriptors'] = train_df['MACCS_Descriptors'].apply(string_to_list)
```

'MACCS_Descriptors'에서 문자열로 표현된 리스트를 정수형 리스트로 변환

```python
train_df= train_df.sort_values(['Outcome'], ascending=True)
```

Outcome 열을 기준으로 오름차순 정렬

Train code-데이터 분할

```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()


outcomes=(np.unique(train_df['Outcome']))
le.fit(list(set(outcomes)))
y = le.transform( train_df['Outcome'] )

print ("Classes                          : ",(outcomes))
print ("Number of compounds in each class  : ",np.unique([len(y[y==smi]) for smi in y]))
print ("Total number of compounds         : ",len(y))

S =train_df['Outcome']
info = {}
for i,cls in enumerate(S.unique()):
    info.update({cls:i})
    S = S.replace(cls,i)
```

```
Classes                          :  ['0.0' '1.0']
Number of compounds in each class  :  [246 257]
Total number of compounds         :  503
```

```python
y= np.int32((S))
x = np.array(list(train_df['MACCS_Descriptors']))
```

y는 train_df['Outcome"] 열을 32비트 정수형으로 변환한 레이블 데이터 배열

x는 MACCS_Descriptors 열에 저장된 분자의 descriptor를 2d 배열로 변환한 것

LabelEncoder: 범주형 데이터를 정수형으로 변환

np.unique(train_df['Outcome'])
: outcome에 있는 고유한 데이터 추출

le.fit(): 고유 클래스를 숫자 순으로 매핑

y=le.transform(train_df['Outcome'])
: Outcome 열에 있는 데이터를 정수로 변환하여 y에 저장

for i, cls in enumerate(S.unipue())
: 클래스의 이름과 값을 정수로 변환

Train code-모델 학습 및 저장

```
#Random Forest
from sklearn.metrics import make_scorer, cohen_kappa_score
from sklearn.model_selection import GridSearchCV, StratifiedShuffleSplit
from sklearn.ensemble import RandomForestClassifier

paramgrid = {
    "max_features": [
        x.shape[1], x.shape[1] // 2, x.shape[1] // 4, x.shape[1] // 12, x.shape[1] // 10,
        x.shape[1] // 7, x.shape[1] // 5, x.shape[1] // 3
    ],
     "n_estimators": [10, 100, 300, 500],}
```

max_feature : 각 트리가 선택할 수 있는 최대 특징 수

n_estimators : 랜덤 포레스트에서 사용할 트리의 개수

```
cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_state=24)
```

-> 데이터를 여러 번 섞어 학습용과 검증용으로 나누는 교차검증 방식

```
kappa_scorer = make_scorer(cohen_kappa_score, weights='quadratic')
```

-> 분류 모델의 성능을 측정하는 지표, 분류 결과가 무작위 추측보다 얼마나 더 나은지를 나타냄

```
model_rf_maccskey = GridSearchCV(estimator=RandomForestClassifier(class_weight='balanced'),
                     param_grid=paramgrid,
                     scoring=kappa_scorer,
                     cv=cv,
                     verbose=1,
                     n_jobs=1)
```

class_weight=‘balanced’ : 클래스의 불균형을 자동으로 보정

```
model_rf_maccskey.fit(x, y)
```

-> 최적의 조합을 찾기 위한 학습

```
best_model_rf_maccskey = model_rf_maccskey.best_estimator_
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
```

```
import joblib
joblib.dump(best_model_rf_maccskey, '/content/drive/MyDrive/Model_acute_dermal_RandomForest_maccskeys.pkl',compress=9)
```

-> 최적의 모델 저장

Train code-모델 학습 및 저장

```python
#XGB
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV, StratifiedShuffleSplit
from sklearn.metrics import make_scorer, cohen_kappa_score

paramgrid = {
     "max_depth": [3, 5, 7, 10],
     "n_estimators": [100, 200, 300],
     "learning_rate": [0.01, 0.1, 0.2]
}

cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_state=24)

kappa_scorer = make_scorer(cohen_kappa_score, weights='quadratic')


model_xgb_maccskey = GridSearchCV(
    estimator=XGBClassifier(use_label_encoder=False, eval_metric='mlogloss'),
    param_grid=paramgrid,
    scoring=kappa_scorer,
    cv=cv,
    verbose=1,
    n_jobs=1
)

model_xgb_maccskey.fit(x, y)

best_model_xgb_maccskey = model_xgb_maccskey.best_estimator_
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
```

max_depth : 트리 최대 깊이
n_estimators : 트리의 개수
Learning_rate : 학습 비율

use_label_encoder=False : 경고메세지 방지 옵션
eval_metric='molgloss' : 다중 클래스 문제에서 로그 손실 함수를 설정

```python
import joblib
joblib.dump(best_model_xgb_maccskey, '/content/drive/MyDrive/Model_acute_dermal_XGBoost_maccskeys.pkl',compress=9)
```

-> 최적의 모델 저장

Train code-모델 학습 및 저장

```python
#SVM
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, StratifiedShuffleSplit
from sklearn.metrics import make_scorer, cohen_kappa_score

paramgrid = {
    "C": [0.1, 1, 10, 100],
    "kernel": ['linear', 'poly', 'rbf', 'sigmoid'],
    "gamma": ['scale', 'auto']
}

cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_state=24)

kappa_scorer = make_scorer(cohen_kappa_score, weights='quadratic')


model_svm_maccskey = GridSearchCV(
    estimator=SVC(probability=True),
    param_grid=paramgrid,
    scoring=kappa_scorer,
    cv=cv,
    verbose=1,
    n_jobs=1
)

model_svm_maccskey.fit(x, y)

best_model_svm_maccskey = model_svm_maccskey.best_estimator_
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
```

c : 정규화된 파라미터로, 모델 복잡도를 조정함
kernel : 커널 함수의 종류 선택
gamma : 커널 계수

- Model_acute_dermal_XGBoost_maccskeys.pkl
- Model_acute_dermal_RandomForest_maccskeys.pkl
- Model_acute_dermal_SupportVectorMachine_maccskeys.pkl
- Model_acute_dermal_XGBoost_morgan.pkl
- Model_acute_dermal_SupportVectorMachine_morgan.pkl
- Model_acute_dermal_RandomForest_morgan.pkl
- Modelo_acute_dermal_RandomForest_modred.pkl
- Modelo_acute_dermal_SupportVectorMachine_modred.pkl
- Modelo_acute_dermal_XGBoost_modred.pkl

-> 총 9개의 모델

```python
import joblib
joblib.dump(best_model_svm_maccskey, '/content/drive/MyDrive/Model_acute_dermal_SupportVectorMachine_maccskeys.pkl',compress=9)
```

-> 최적의 모델 저장

Train code-모델 평가

```python
# Random Forest train score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix, precision_score, recall_score, f1_score
import numpy as np

seed = 42
np.random.seed(seed)
n_splits = 5
cv = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=seed)

train_accuracies, train_auc_scores, train_precisions, train_recalls, train_f1_scores = [], [], [], [], []
train_specificities, train_sensitivity_scores, train_ppvs, train_npvs, train_ccrs = [], [], [], [], []
test_accuracies, test_auc_scores, test_precisions, test_recalls, test_f1_scores = [], [], [], [], []
test_specificities, test_sensitivity_scores, test_ppvs, test_npvs, test_ccrs = [], [], [], [], []
confusion_matrices = []

def calculate_ccr(sensitivity, specificity):
    return (sensitivity + specificity) / 2
```

```
Confusion Matrix for Fold 1:
[[40 12]
 [13 36]]
Confusion Matrix for Fold 2:
[[39 13]
 [12 37]]
Confusion Matrix for Fold 3:
[[38 13]
 [16 34]]
Confusion Matrix for Fold 4:
[[35 16]
 [12 37]]
Confusion Matrix for Fold 5:
[[44  7]
 [13 36]]
```

```
Test Performance:
Overall Test CV Accuracy: 0.7475643564356436
Overall Test CV AUC: 0.8316693754424846
Overall Test CV Precision: 0.7497453530383801
Overall Test CV Recall (Sensitivity): 0.7319183673469387
Overall Test CV F1 Score: 0.7397745216555036
Overall Test CV Specificity: 0.7626696832579186
Overall Test CV PPV (Positive Predictive Value): 0.7497453530383801
Overall Test CV NPV (Negative Predictive Value): 0.7479474485627907
Overall Test CV CCR (Correct Classification Rate): 0.7472940253024287

Train Performance:
Overall Train CV Accuracy: 0.9970186289396687
Overall Train CV AUC: 0.9999357080952971
Overall Train CV Precision: 0.9939545700609905
Overall Train CV Recall (Sensitivity): 1.0
Overall Train CV F1 Score: 0.9969645630134565
Overall Train CV Specificity: 0.9941652853421739
Overall Train CV PPV (Positive Predictive Value): 0.9939545700609905
Overall Train CV NPV (Negative Predictive Value): 1.0
Overall Train CV CCR (Correct Classification Rate): 0.997082642671087
```

```python
for train_index, test_index in cv.split(x, y):
    X_train, X_test = x[train_index], x[test_index]
    y_train, y_test = y[train_index], y[test_index]

    best_model_rf_maccskey.fit(X_train, y_train)

    y_train_pred = best_model_rf_maccskey.predict(X_train)
    y_train_proba = best_model_rf_maccskey.predict_proba(X_train)[:, 1]
    y_test_pred = best_model_rf_maccskey.predict(X_test)
    y_test_proba = best_model_rf_maccskey.predict_proba(X_test)[:, 1]

    train_accuracies.append(accuracy_score(y_train, y_train_pred))
    train_auc_scores.append(roc_auc_score(y_train, y_train_proba))
    train_precisions.append(precision_score(y_train, y_train_pred, zero_division=0))
    train_recalls.append(recall_score(y_train, y_train_pred))
    train_f1_scores.append(f1_score(y_train, y_train_pred))
    train_cm = confusion_matrix(y_train, y_train_pred)
    tn_train, fp_train, fn_train, tp_train = train_cm.ravel()
    train_sensitivity = tp_train / (tp_train + fn_train)
    train_specificity = tn_train / (tn_train + fp_train)
    train_sensitivity_scores.append(train_sensitivity)
    train_specificities.append(train_specificity)
    train_ppv = tp_train / (tp_train + fp_train) if (tp_train + fp_train) > 0 else 0
    train_npv = tn_train / (tn_train + fn_train) if (tn_train + fn_train) > 0 else 0
    train_ppvs.append(train_ppv)
    train_npvs.append(train_npv)
    train_ccr = calculate_ccr(train_sensitivity, train_specificity)
    train_ccrs.append(train_ccr)

    test_accuracies.append(accuracy_score(y_test, y_test_pred))
    test_auc_scores.append(roc_auc_score(y_test, y_test_proba))
    test_precisions.append(precision_score(y_test, y_test_pred, zero_division=0))
    test_recalls.append(recall_score(y_test, y_test_pred))
    test_f1_scores.append(f1_score(y_test, y_test_pred))
    cm = confusion_matrix(y_test, y_test_pred)
    confusion_matrices.append(cm)
    tn, fp, fn, tp = cm.ravel()
    test_sensitivity = tp / (tp + fn)
    test_specificity = tn / (tn + fp)
    test_sensitivity_scores.append(test_sensitivity)
    test_specificities.append(test_specificity)
    test_ppv = tp / (tp + fp) if (tp + fp) > 0 else 0
    test_npv = tn / (tn + fn) if (tn + fn) > 0 else 0
    test_ppvs.append(test_ppv)
    test_npvs.append(test_npv)
    test_ccr = calculate_ccr(test_sensitivity, test_specificity)
    test_ccrs.append(test_ccr)
```

Test code-데이터 로드, 분할

```python
file="/content/drive/MyDrive/test_set_acute_dermal_features_rdkitcdk.sdf"
test_df = load_sdf_to_df(file)
```

```python
def string_to_list(bit_string):
    if isinstance(bit_string, str):
        return list(map(int, bit_string.strip('[]').split(', ')))
    else:
        return bit_string

test_df['Morgan_Descriptors'] = test_df['Morgan_Descriptors'].apply(string_to_list)
test_df['MACCS_Descriptors'] = test_df['MACCS_Descriptors'].apply(string_to_list)

def string_to_list(descriptor):
    if isinstance(descriptor, str):
        return list(map(float, descriptor.strip('[]').split(',')))
    return descriptor

test_df['Modred_Descriptor'] = test_df['Modred_Descriptor'].apply(string_to_list)
```

```python
from sklearn.preprocessing import LabelEncoder
import numpy as np

# Create the label encoder
le = LabelEncoder()

# Get unique outcomes
outcomes = np.unique(test_df['Outcome'])
le.fit(list(set(outcomes)))

# Transform the 'Outcome' column
y = le.transform(test_df['Outcome'])

# Print class information
print("Classes                          : ", outcomes)
print("Number of cpds in each class     : ", [len(y[y == smi]) for smi in np.unique(y)])
print("Total number of cpds             : ", len(y))

# Explicitly map the outcome classes
S = test_df['Outcome']

# Ensure the class mapping is applied explicitly and consistently
info = {0: 0, 1: 1}  # Ensure '0' is mapped to 0 and '1' is mapped to 1
S = S.replace(info)

# Print the class replacement info for reference
print("Class mapping: ", info)
```

```
Classes                      :  ['0.0' '1.0']
Number of cpds in each class :  [57, 69]
Total number of cpds         :  126
Class mapping:  {0: 0, 1: 1}
```

Test code-학습된 모델 로드, Consensus

```python
import joblib
rf_morgan = joblib.load('/content/drive/MyDrive/Model_acute_dermal_RandomForest_morgan.pkl')
rf_maccs = joblib.load('/content/drive/MyDrive/Model_acute_dermal_RandomForest_maccskeys.pkl')
rf_modred = joblib.load('/content/drive/MyDrive/Modelo_acute_dermal_RandomForest_modred.pkl')
```

```python
y_true = test_df['Outcome'].astype(int)
```
-> 실제 정답 레이블

```python
def predict_with_models_probabilities(moldf):
    morgan_probs = rf_morgan.predict_proba(np.array(list(test_df['Morgan_Descriptors'].values)))[:, 1]
    maccs_probs = rf_maccs.predict_proba(np.array(list(test_df['MACCS_Descriptors'].values)))[:, 1]
    modred_probs = rf_modred.predict_proba(np.array(list(test_df['Modred_Descriptor'].values)))[:, 1]

    mean_probs = np.mean([morgan_probs, maccs_probs, modred_probs], axis=0)
  # mean_probs = np.mean([morgan_probs, maccs_probs], axis=0)
  # mean_probs = np.mean([morgan_probs, modred_probs], axis=0)

    final_predictions = (mean_probs > 0.5).astype(int)

    return final_predictions, mean_probs
```

1. 개별 모델의 확률 예측
2. 모든 모델의 확률의 평균
3. 평균 확률 값을 기준으로 클래스 결정

```python
final_predictions, mean_probs = predict_with_models_probabilities(test_df)
test_df['Predictions'] = final_predictions
test_df['Mean_Probabilities'] = mean_probs
```

```python
from sklearn.metrics import confusion_matrix, accuracy_score, roc_auc_score, f1_score

conf_matrix = confusion_matrix(y_true, final_predictions)
print("Confusion Matrix:")
print(conf_matrix)

accuracy = accuracy_score(y_true, final_predictions)
print("Accuracy:", accuracy)

auc_score = roc_auc_score(y_true, mean_probs)
print("AUC Score:", auc_score)

f1 = f1_score(y_true, final_predictions, average='binary')
print("F1 Score:", f1)

tn, fp, fn, tp = conf_matrix.ravel()
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)

ccr = (sensitivity + specificity) / 2
print("CCR (Correct Classification Rate):", ccr)

ppv = tp / (tp + fp)
print("PPV (Positive Predictive Value):", ppv)

npv = tn / (tn + fn)
print("NPV (Negative Predictive Value):", npv)
```

	SMILES	Predictions	Mean_Probabilities
0	CCSC(=O)N(CC)C1CCCCC1	0	0.179754
1	CNCC(O)c1ccc(O)c(O)c1	0	0.329579
2	CC1CC(=O)C=C2CCC3C4CCC(O)C4(C)CCC3C21C	0	0.068781
3	CN(C)N	1	0.775626
4	CC1OC(C)OC(C)OC(C)O1	0	0.348168
..	...	...	...
121	CNC(=O)N(C)c1nnc(C(F)(F)F)s1	0	0.174134
122	CC1(C)CCC(=Cc2ccc(Cl)cc2)C1(O)Cn1cncn1	0	0.225792
123	Sc1ccccc1	1	0.833504
124	COc1cc(OC)nc(NC(=O)Oc2ccccc2)n1	0	0.163199
125	COc1cc(C(F)(F)F)ccn1	0	0.372882

```
Confusion Matrix:
[[44 13]
 [18 51]]
Accuracy: 0.753968253968254
AUC Score: 0.844902110348346
F1 Score: 0.7669172932330827
Sensitivity: 0.7391304347826086
Specificity: 0.7719298245614035
CCR (Correct Classification Rate): 0.7555301296720061
PPV (Positive Predictive Value): 0.796875
NPV (Negative Predictive Value): 0.7096774193548387
```

Test code-Individual Performance

```
import numpy as np
from sklearn.metrics import confusion_matrix, accuracy_score, roc_auc_score, f1_score
#RF RF RF
def predict_with_models_probabilities(test_df):
    morgan_probs = rf_morgan.predict_proba(np.array(list(test_df['Morgan_Descriptors'].values)))[:, 1]
    maccs_probs = rf_maccs.predict_proba(np.array(list(test_df['MACCS_Descriptors'].values)))[:, 1]
    modred_probs = rf_modred.predict_proba(np.array(list(test_df['Modred_Descriptor'].values)))[:, 1]

    morgan_predictions = (morgan_probs > 0.5).astype(int)
    maccs_predictions = (maccs_probs > 0.5).astype(int)
    modred_predictions = (modred_probs > 0.5).astype(int)

    return morgan_predictions, morgan_probs, maccs_predictions, maccs_probs, modred_predictions, modred_probs

def evaluate_performance(y_true, predictions, probabilities):
    conf_matrix = confusion_matrix(y_true, predictions)
    accuracy = accuracy_score(y_true, predictions)
    auc_score = roc_auc_score(y_true, probabilities)
    f1 = f1_score(y_true, predictions, average='binary')

    tn, fp, fn, tp = conf_matrix.ravel()
    sensitivity = tp / (tp + fn)
    specificity = tn / (tn + fp)
    ccr = (sensitivity + specificity) / 2
    ppv = tp / (tp + fp)
    npv = tn / (tn + fn)

    return {
        "Confusion Matrix": conf_matrix,
        "Accuracy": accuracy,
        "AUC Score": auc_score,
        "F1 Score": f1,
        "Sensitivity": sensitivity,
        "Specificity": specificity,
        "CCR (Correct Classification Rate)": ccr,
        "PPV (Positive Predictive Value)": ppv,
        "NPV (Negative Predictive Value)": npv
    }


morgan_predictions, morgan_probs, maccs_predictions, maccs_probs, modred_predictions, modred_probs = predict_with_models_probabilities(test_df)

morgan_performance = evaluate_performance(y_true, morgan_predictions, morgan_probs)
maccs_performance = evaluate_performance(y_true, maccs_predictions, maccs_probs)
modred_performance = evaluate_performance(y_true, modred_predictions, modred_probs)
```

각 모델을 사용하여 클래스 1에 대한 predict_proba 계산

각 확률을 기준으로 0.5 이상의 값을 클래스 1, 그렇지 않으면 클래스 0으로 변환해 최종 예측 값을 생성

y_true : 실제 라벨 값
predictions : 모델의 이진 예측 값
probabilities : 모델의 클래스 1 확률 값

-> 모델의 성능 평가

Thank you