ML_pipeline

November 24, 2019

[1]:
```
# This script trains and select models for this problem
# load the packages
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.metrics import make_scorer
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
import matplotlib
```

[2]:
```
df = pd.read_csv('../data/processed_sleep_fft.csv')
cat_ftrs = ['channel_name']
scalar_ftrs = ['alpha', 'theta', 'slowwave', 'sigma']
```

[3]:
```
le = LabelEncoder()
y = le.fit_transform(df['label'])
subject_ID = df['subject']
nap_ID = df['NAP']
dropc = ['Unnamed: 0', 'label', 'subject', 'NAP']
X = df.drop(columns= dropc)
# check balance
classes, counts = np.unique(y, return_counts=True)
for i in range(len(classes)):
    print ('balance', i, counts[i]/ len(y))
```

```
balance 0 0.1071059175634339
balance 1 0.1365286855482934
balance 2 0.09823727470786295
balance 3 0.34462269756387404
balance 4 0.31350542461653574
```

[4]:
```
# encode groups 4 subject ID * 2 naps
import itertools
n_subject = np.unique(subject_ID)
n_nap = np.unique(nap_ID)
iterset = (list(itertools.product(n_subject, n_nap)))
group = np.zeros((len(subject_ID), 1))
i = 0
for sbj, nap in iterset:
    idx = np.logical_and(subject_ID == sbj, nap_ID== nap)
    group[idx] = i
    i+= 1
```

[5]:
```
# from sklearn.model_selection import GroupKFold
# from sklearn.model_selection import GroupShuffleSplit
# def
 ↪ML_pipeline_groups_GridSearchCV_SVC(X,y,groups,random_state,n_folds):
 ↪
#     # create a test set based on groups
#     splitter = GroupShuffleSplit(n_splits=1,test_size=0.
 ↪2,random_state=random_state)
#     for i_other,i_test in splitter.split(X, y, groups):
#         X_other, y_other, groups_other = X.iloc[i_other],
 ↪y[i_other], groups[i_other]
#         X_test, y_test, groups_test = X.iloc[i_test], y[i_test],
 ↪groups[i_test]
#     # splitter for _other
#     kf = GroupKFold(n_splits=n_folds)
#     # create the pipeline: preprocessor + supervised ML method
#     cat_ftrs = ['channel_name']
#     cont_ftrs = ['alpha', 'theta', 'slowwave', 'sigma']
#     cat_transformer = Pipeline(steps = [
#         ('imputer1', SimpleImputer(missing_values='0.0',
 ↪strategy='constant',fill_value='missing')),
#         ('onehot', OneHotEncoder(sparse = False, categories =
 ↪'auto'))])
#     cont_transformer = Pipeline(steps = [
#         ('imputer2', SimpleImputer(missing_values = np.nan,strategy
 ↪= 'mean')),
#         ('scaler', StandardScaler())])
#     preprocessor = ColumnTransformer(remainder='passthrough',
#     transformers=[
```

```
#         ('num', cont_transformer, cont_ftrs),
#         ('cat', cat_transformer, cat_ftrs)])

#     # make overall pipeline
#     pipe = make_pipeline(
#         preprocessor,
#         SVC(probability = True, max_iter = 1000))

#     # the parameter(s) we want to tune
#     param_grid = {'svc__C': np.logspace(-2,2,num=5),'svc__gamma':
  ↪np.logspace(-2,2,num=5)}
#     # prepare gridsearch
#     grid = GridSearchCV(pipe, param_grid=param_grid,scoring =
  ↪make_scorer(accuracy_score),
#                         cv=kf, return_train_score = True,iid=True,
  ↪n_jobs = -1)
#     # do kfold CV on _other
#     grid.fit(X_other, y_other, groups_other)
#     return grid, grid.score(X_test, y_test)
```

[ ]:
```
# test_scores_SVC = []
# for i in range(5):
#     print('Starting:', i, 'process')
#     grid, test_score =
  ↪ML_pipeline_groups_GridSearchCV_SVC(X,y,group,i*42,2)
#     print(grid.best_params_)
#     print('best CV score:',grid.best_score_)
#     print('test score:',test_score)
#     test_scores_SVC.append(test_score)
# print('test accuracy:',np.around(np.mean(test_scores_SVC),2),'+/
  ↪-',np.around(np.std(test_scores_SVC),2))
```

```
Starting: 0 process
```

[5]:
```
from sklearn.model_selection import GroupKFold
from sklearn.model_selection import GroupShuffleSplit
def
  ↪ML_pipeline_groups_GridSearchCV_Logistic(X,y,groups,random_state,n_folds):
  ↪
    # create a test set based on groups
    splitter = GroupShuffleSplit(n_splits=1,test_size=0.
  ↪2,random_state=random_state)
    for i_other,i_test in splitter.split(X, y, groups):
        X_other, y_other, groups_other = X.iloc[i_other], y[i_other],
  ↪groups[i_other]
        X_test, y_test, groups_test = X.iloc[i_test], y[i_test],
  ↪groups[i_test]
```

```
    # splitter for _other
    kf = GroupKFold(n_splits=n_folds)
    # create the pipeline: preprocessor + supervised ML method
    cat_ftrs = ['channel_name']
    cont_ftrs = ['alpha', 'theta', 'slowwave', 'sigma']
    cat_transformer = Pipeline(steps = [
        ('imputer1', SimpleImputer(missing_values='0.0',
 ↪strategy='constant',fill_value='missing')),
        ('onehot', OneHotEncoder(sparse = False, categories =
 ↪'auto'))])
    cont_transformer = Pipeline(steps = [
        ('imputer2', SimpleImputer(missing_values = np.nan,strategy =
 ↪'mean')),
        ('scaler', StandardScaler())])
    preprocessor = ColumnTransformer(remainder='passthrough',
    transformers=[
        ('num', cont_transformer, cont_ftrs),
        ('cat', cat_transformer, cat_ftrs)])

    # make overall pipeline
    pipe = make_pipeline(
        preprocessor,
        LogisticRegression(penalty = 'l1', solver = 'saga', max_iter
 ↪= 1000, multi_class = 'multinomial'))

    # the parameter(s) we want to tune
    param_grid = {'logisticregression__C': np.logspace(-2,2,num=5)}
    # prepare gridsearch
    grid = GridSearchCV(pipe, param_grid=param_grid,scoring =
 ↪make_scorer(accuracy_score),
                        cv=kf, return_train_score = True,iid=True,
 ↪n_jobs = -1)
    # do kfold CV on _other
    grid.fit(X_other, y_other, groups_other)
    return grid, grid.score(X_test, y_test)
```

[7]:
```
test_scores_logistic = []
for i in range(5):
    print('Starting:', i, 'process')
    grid, test_score =
 ↪ML_pipeline_groups_GridSearchCV_Logistic(X,y,group,i*42,2)
    print(grid.best_params_)
    print('best CV score:',grid.best_score_)
    print('test score:',test_score)
    test_scores_logistic.append(test_score)
```

```
print('test accuracy:',np.around(np.mean(test_scores_logistic),2),'+/
  ↪-',np.around(np.std(test_scores_logistic),2))
```

```
Starting: 0 process
{'logisticregression__C': 1.0}
best CV score: 0.6240503012837306
test score: 0.7443181818181818
Starting: 1 process
{'logisticregression__C': 0.01}
best CV score: 0.6500014695940981
test score: 0.5604308985811876
Starting: 2 process
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/sag.py:337:
ConvergenceWarning: The max_iter was reached which means the coef_ did
  ↪not
converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
{'logisticregression__C': 10.0}
best CV score: 0.6702455337498902
test score: 0.6205954897815363
Starting: 3 process
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/sag.py:337:
ConvergenceWarning: The max_iter was reached which means the coef_ did
  ↪not
converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
{'logisticregression__C': 100.0}
best CV score: 0.6689908796704913
test score: 0.5892061828661252
Starting: 4 process
{'logisticregression__C': 100.0}
best CV score: 0.6308379625547423
test score: 0.6586967945349448
test accuracy: 0.63 +/- 0.06
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/sag.py:337:
ConvergenceWarning: The max_iter was reached which means the coef_ did
  ↪not
converge
  "the coef_ did not converge", ConvergenceWarning)
```

[5]:
```
from sklearn.model_selection import GroupKFold
from sklearn.model_selection import GroupShuffleSplit
```

```
def ML_pipeline_groups_GridSearchCV_RandomForest(X,y,groups,random_state,n_folds):
    # create a test set based on groups
    splitter = GroupShuffleSplit(n_splits=1,test_size=0.2,random_state=random_state)
    for i_other,i_test in splitter.split(X, y, groups):
        X_other, y_other, groups_other = X.iloc[i_other], y[i_other],groups[i_other]
        X_test, y_test, groups_test = X.iloc[i_test], y[i_test],groups[i_test]
    # splitter for _other
    kf = GroupKFold(n_splits=n_folds)
    # create the pipeline: preprocessor + supervised ML method
    cat_ftrs = ['channel_name']
    cont_ftrs = ['alpha', 'theta', 'slowwave', 'sigma']
    cat_transformer = Pipeline(steps = [
        ('imputer1', SimpleImputer(missing_values='0.0',strategy='constant',fill_value='missing')),
        ('onehot', OneHotEncoder(sparse = False, categories ='auto'))])
    cont_transformer = Pipeline(steps = [
        ('imputer2', SimpleImputer(missing_values = np.nan,strategy ='mean')),
        ('scaler', StandardScaler())])
    preprocessor = ColumnTransformer(remainder='passthrough',
    transformers=[
        ('num', cont_transformer, cont_ftrs),
        ('cat', cat_transformer, cat_ftrs)])

    # make overall pipeline
    pipe = make_pipeline(
        preprocessor,
        RandomForestClassifier(random_state= random_state))

    # specify parameters
    param_grid = {'randomforestclassifier__max_depth' : np.logspace(0,3, num=5),
                  'randomforestclassifier__n_estimators' : np.linspace(1, 100, num = 5, dtype = int)}
    # prepare gridsearch
    grid = GridSearchCV(pipe, param_grid=param_grid,scoring =make_scorer(accuracy_score),
                        cv=kf, return_train_score = True,iid=True,n_jobs = -1)
    # do kfold CV on _other
```

```
    grid.fit(X_other, y_other, groups_other)
    return grid, grid.score(X_test, y_test)
```

[7]:
```
test_scores_randomForest = []
for i in range(5):
    print('Starting:', i, 'process')
    grid, test_score =
  ↪ML_pipeline_groups_GridSearchCV_RandomForest(X,y,group,i*42,2)
    print(grid.best_params_)
    print('best CV score:',grid.best_score_)
    print('test score:',test_score)
    test_scores_randomForest.append(test_score)
print('test accuracy:',np.around(np.
  ↪mean(test_scores_randomForest),2),'+/-',np.around(np.
  ↪std(test_scores_randomForest),2))
```

```
Starting: 0 process
{'randomforestclassifier__max_depth': 31.622776601683793,
'randomforestclassifier__n_estimators': 100}
best CV score: 0.6520536779902775
test score: 0.738546176046176
Starting: 1 process
{'randomforestclassifier__max_depth': 31.622776601683793,
'randomforestclassifier__n_estimators': 75}
best CV score: 0.6450048496605237
test score: 0.6138553161674549
Starting: 2 process
/opt/conda/lib/python3.7/site-
packages/joblib/externals/loky/process_executor.py:706: UserWarning: A
  ↪worker
stopped while some jobs were given to the executor. This can be caused
  ↪by a too
short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
{'randomforestclassifier__max_depth': 177.82794100389228,
'randomforestclassifier__n_estimators': 75}
best CV score: 0.6883158790225586
test score: 0.6526603241719521
Starting: 3 process
/opt/conda/lib/python3.7/site-
packages/joblib/externals/loky/process_executor.py:706: UserWarning: A
  ↪worker
stopped while some jobs were given to the executor. This can be caused
  ↪by a too
short worker timeout or by a memory leak.
```

```
  "timeout or by a memory leak.", UserWarning
```

```
{'randomforestclassifier__max_depth': 31.622776601683793,
'randomforestclassifier__n_estimators': 100}
best CV score: 0.671726978523095
test score: 0.643611911623439
Starting: 4 process
{'randomforestclassifier__max_depth': 31.622776601683793,
'randomforestclassifier__n_estimators': 50}
best CV score: 0.6694295035711136
test score: 0.6779646172709757
test accuracy: 0.67 +/- 0.04
```

[8]:
```
from sklearn.model_selection import GroupKFold
from sklearn.model_selection import GroupShuffleSplit
from sklearn.model_selection import ParameterGrid
from xgboost import XGBClassifier
def
 ↪ML_pipeline_groups_GridSearchCV_XGboost(X,y,groups,random_state,n_folds):
 ↪
    # create a test set based on groups
    splitter = GroupShuffleSplit(n_splits=1,test_size=0.
 ↪2,random_state=random_state)
    for i_other,i_test in splitter.split(X, y, groups):
        X_other, y_other, groups_other = X.iloc[i_other], y[i_other],
 ↪groups[i_other]
        X_test, y_test, groups_test = X.iloc[i_test], y[i_test],
 ↪groups[i_test]
    # splitter for _other
    kf = GroupKFold(n_splits=n_folds)
    # create the pipeline: preprocessor + supervised ML method
    cat_ftrs = ['channel_name']
    cont_ftrs = ['alpha', 'theta', 'slowwave', 'sigma']
    cat_transformer = Pipeline(steps = [
        ('imputer1', SimpleImputer(missing_values='0.0',
 ↪strategy='constant',fill_value='missing')),
        ('onehot', OneHotEncoder(sparse = False, categories =
 ↪'auto'))])
    cont_transformer = Pipeline(steps = [
        ('scaler', StandardScaler())])
    preprocessor = ColumnTransformer(remainder='passthrough',
    transformers=[
        ('num', cont_transformer, cont_ftrs),
        ('cat', cat_transformer, cat_ftrs)])

    # make overall pipeline
    pipe = make_pipeline(
```

```
        preprocessor,
        XGBClassifier(seed = random_state))

    # specify parameters
    param_grid = {"xgbclassifier__reg_alpha":np.logspace(-2,2,num=5) }
    # prepare gridsearch
    grid = GridSearchCV(pipe, param_grid=(param_grid),scoring =
  ↪make_scorer(accuracy_score),
                        cv=kf, return_train_score = True,iid=True,
  ↪n_jobs = -1)
    # do kfold CV on _other
    grid.fit(X_other, y_other, groups_other)
    return grid, grid.score(X_test, y_test)
```

[10]:
```
test_scores_xgboost = []
for i in range(5):
    print('Starting:', i, 'process')
    grid, test_score =
  ↪ML_pipeline_groups_GridSearchCV_XGboost(X,y,group,i*42,2)
    print(grid.best_params_)
    print('best CV score:',grid.best_score_)
    print('test score:',test_score)
    test_scores_xgboost.append(test_score)
print('test accuracy:',np.around(np.mean(test_scores_xgboost),2),'+/
  ↪-',np.around(np.std(test_scores_xgboost),2))
```

```
Starting: 0 process
{'xgbclassifier__reg_alpha': 1.0}
best CV score: 0.6285622798591098
test score: 0.7369227994227994
Starting: 1 process
{'xgbclassifier__reg_alpha': 0.01}
best CV score: 0.6525291714428475
test score: 0.5638465580662112
Starting: 2 process
{'xgbclassifier__reg_alpha': 0.1}
best CV score: 0.6726510017894335
test score: 0.6375088090204369
Starting: 3 process
{'xgbclassifier__reg_alpha': 1.0}
best CV score: 0.6698440717858194
test score: 0.5871102960440137
Starting: 4 process
{'xgbclassifier__reg_alpha': 0.1}
best CV score: 0.6447109308409017
test score: 0.6680679628656507
test accuracy: 0.64 +/- 0.06
```

[12]:
```
test_scores_xgboost_2 = []
for i in range(5):
    print('Starting:', i, 'process')
    grid, test_score =
  ↪ML_pipeline_groups_GridSearchCV_XGboost(X,y,group,i*42,5)
    print(grid.best_params_)
    print('best CV score:',grid.best_score_)
    print('test score:',test_score)
    test_scores_xgboost_2.append(test_score)
print('test accuracy:',np.around(np.mean(test_scores_xgboost_2),2),'+/
  ↪-',np.around(np.std(test_scores_xgboost_2),2))
```

```
Starting: 0 process
{'xgbclassifier__reg_alpha': 0.01}
best CV score: 0.6231770151078508
test score: 0.73502886002886
Starting: 1 process
{'xgbclassifier__reg_alpha': 0.01}
best CV score: 0.6798930135496576
test score: 0.5638465580662112
Starting: 2 process
{'xgbclassifier__reg_alpha': 0.01}
best CV score: 0.6609463463287277
test score: 0.6373326286116984
Starting: 3 process
{'xgbclassifier__reg_alpha': 0.1}
best CV score: 0.6652544866137099
test score: 0.5896428259540651
Starting: 4 process
{'xgbclassifier__reg_alpha': 0.01}
best CV score: 0.6492960644270053
test score: 0.6692065160273253
test accuracy: 0.64 +/- 0.06
```

[ ]: