FeatureImportance

November 24, 2019

[3]:
```
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
```

[4]:
```
df = pd.read_csv('../data/processed_sleep_fft.csv')
cat_ftrs = ['channel_name']
scalar_ftrs = ['alpha', 'theta', 'slowwave', 'sigma']
```

[5]:
```
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)
ftr_names = X.columns
# 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
```

[6]:
```
# 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
```

[7]:
```
# Run the best model and save the result on hub
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":[1.0] }
    # 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), X_test, y_test
```

[8]:
```
grid, test_score, X_test, y_test =
  ↪ML_pipeline_groups_GridSearchCV_XGboost(X,y,group,0,5)
print(grid.best_params_)
print('best CV score:',grid.best_score_)
print('test score:',test_score)
```

```
{'xgbclassifier__reg_alpha': 1.0}
best CV score: 0.622070852618403
test score: 0.7369227994227994
```

[9]:
```
import pickle
import os

filename = '../results/grid.save'
file = open(filename, 'wb')
pickle.dump((grid,X_test,y_test),file)
file.close()
```

[ ]:
```
# import pickle
# file = open('../results/grid.save', 'rb')
# a, b, c = pickle.load(file)
# file.close()
```

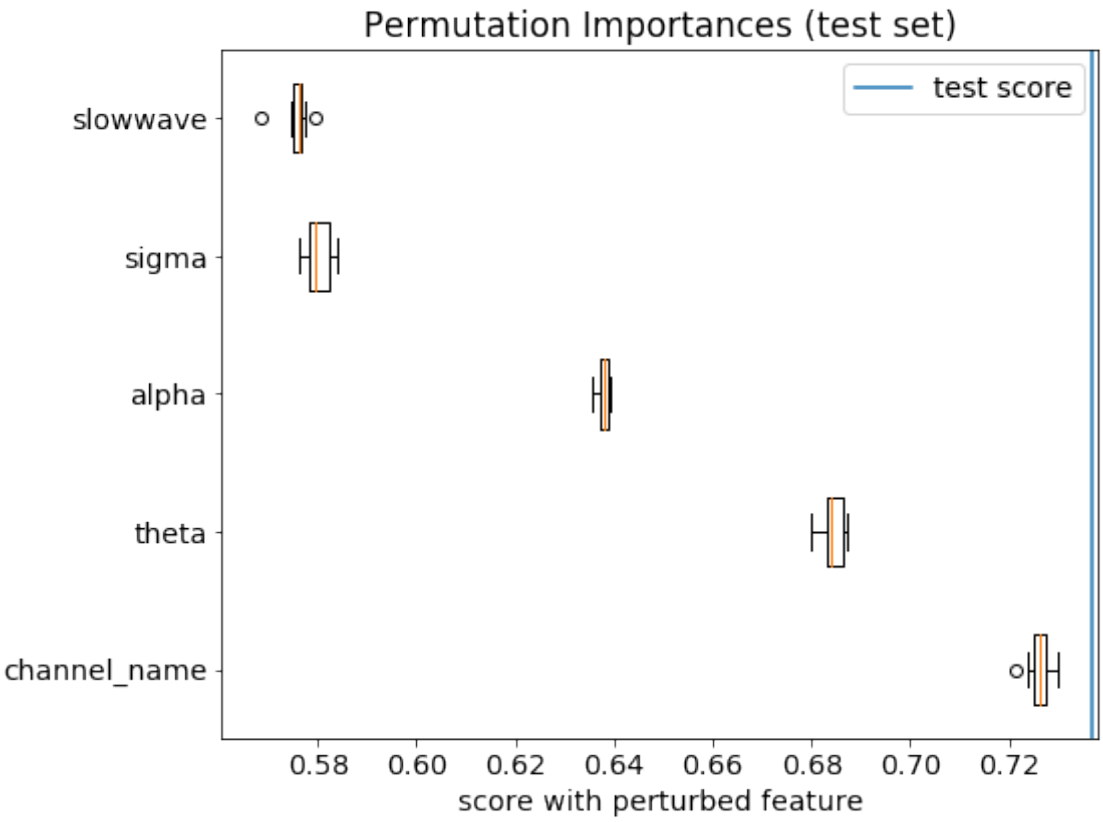
[10]:
```
# Run the permutation


nr_runs = 10
scores = np.zeros([len(ftr_names),nr_runs])
```

```
test_score = grid.score(X_test,y_test)
print('test score = ',test_score)
print('test baseline = ',np.sum(y_test == 0)/len(y_test))
# loop through the features
for i in range(len(ftr_names)):
    print('shuffling '+str(ftr_names[i]))
    acc_scores = []
    for j in range(nr_runs):
        X_test_shuffled = X_test.copy()
        X_test_shuffled[ftr_names[i]] = np.random.
  ↪permutation(X_test[ftr_names[i]].values)
        acc_scores.append(grid.score(X_test_shuffled,y_test))
    print('   shuffled test score:',np.around(np.
  ↪mean(acc_scores),3),'+/-',np.around(np.std(acc_scores),3))
    scores[i] = acc_scores
```

```
test score =  0.7369227994227994
test baseline =  0.08531746031746032
shuffling alpha
   shuffled test score: 0.638 +/- 0.001
shuffling theta
   shuffled test score: 0.684 +/- 0.002
shuffling slowwave
   shuffled test score: 0.576 +/- 0.003
shuffling sigma
   shuffled test score: 0.58 +/- 0.003
shuffling channel_name
   shuffled test score: 0.726 +/- 0.002
```

[12]:
```
sorted_indcs = np.argsort(np.mean(scores,axis=1))[::-1]
plt.rcParams.update({'font.size': 14})
plt.figure(figsize=(8,6))
plt.boxplot(scores[sorted_indcs].
  ↪T,labels=ftr_names[sorted_indcs],vert=False)
plt.axvline(test_score,label='test score')
plt.title("Permutation Importances (test set)")
plt.xlabel('score with perturbed feature')
plt.legend()
plt.tight_layout()
plt.savefig('../figures/feature_permutation.png')
plt.show()
```


Permutation Importances (test set)
test score
slowwave
sigma
alpha
theta
channel_name
0.58
0.60
0.62
0.64
0.66
0.68
0.70
0.72
score with perturbed feature

[ ]: