MLP_Classification

February 8, 2021

[ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.pipeline import make_pipeline
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import glob
import time
```

[ ]:
```
#Get and prepare all the data
aggr_files = []
normal_files = []
for i in range(1,5):
    path_aggr = r'EMG Physical Action Data Set/sub'+str(i)+'/Aggressive/txt'
    aggr_files.extend(glob.glob(os.path.join(path_aggr, "*.txt")))
    path_normal = r'EMG Physical Action Data Set/sub'+str(i)+'/Normal/txt'
    normal_files.extend(glob.glob(os.path.join(path_normal, "*.txt")))

df_from_each_aggr_file = (pd.read_table(f, names=list(range(8))) for f in␣
 ↪aggr_files)
aggr_df = pd.concat(df_from_each_aggr_file, ignore_index=True)

df_from_each_normal_file = (pd.read_table(f, names=list(range(8))) for f in␣
 ↪normal_files)
normal_df = pd.concat(df_from_each_normal_file, ignore_index=True)

aggr_df.insert(8, 'label', 0)
normal_df.insert(8, 'label', 1)

data = pd.concat([aggr_df,normal_df], ignore_index=True)
data = data.dropna()

feature_list = list(range(8))
```

[ ]:
```
#Functions which calculates and prints different performance metrics
def MetricsOfReliability(clf, data_test_df, data_label_df, clf_predict_df):

    tn, fp, fn, tp = metrics.confusion_matrix(data_label_df, clf_predict_df).
 ↪ravel()

    accuracy = (tp + tn)/(tp + fp + tn + fn)
    print('accuracy = ' + str(accuracy))

    # sensitivity = tn/(tn + fp)
    # print('sensitivity = ' + str(sensitivity))

    # specificity = tp/(tp + fn)
    # print('specificity = ' + str(specificity))

    # metrics.plot_roc_curve(clf, data_test_df, data_label_df)
```

[ ]:
```
#perform train-test-split
train, test = train_test_split(data, test_size=0.01, train_size=0.04)
```

[ ]:
```
#Testing different architectures of the MLP and measuring accuracy and compute␣
 ↪time (Without standardizing the data)
hidden_layers = [(50), (100), (200), (500), (50,50), (100,100), (200,200),␣
 ↪(50,50,50), (100,100,100), (200,200,200)]
for hidden_layers in hidden_layers:
    tstart = time.time()
    mlp = MLPClassifier(activation='logistic',␣
 ↪hidden_layer_sizes=hidden_layers, max_iter=800, random_state=1)
    mlp.fit(train[feature_list], train['label'])
    mlp_predict = mlp.predict(test[feature_list])
    tend = time.time()

    print('\n' + str(hidden_layers) + ':')
    MetricsOfReliability(mlp, test[feature_list], test['label'], mlp_predict)
    print('time: ' + str(tend - tstart))
```

[ ]:
```
#Testing different architectures of the MLP and measuring accuracy and compute␣
 ↪time (With standardizing the data)
scaler = StandardScaler()
train_tf = scaler.fit_transform(train[feature_list])
test_tf = scaler.transform(test[feature_list])

hidden_layers = [(50), (100), (200), (500), (50,50), (100,100), (200,200),␣
 ↪(50,50,50), (100,100,100), (200,200,200)]
for hidden_layers in hidden_layers:
    tstart = time.time()
```

```
    mlp = MLPClassifier(activation='logistic',␣
↪hidden_layer_sizes=hidden_layers, max_iter=800, random_state=1)
    mlp.fit(train_tf, train['label'])
    mlp_predict = mlp.predict(test_tf)
    tend = time.time()

    print('\n' + str(hidden_layers) + ':')
    MetricsOfReliability(mlp, test_tf, test['label'], mlp_predict)
    print('time: ' + str(tend - tstart))
```

[ ]:
```
#Measuring accuracy and compute time for the SVM with rbf kernel
tstart = time.time()
svm_rbf = make_pipeline(StandardScaler(), SVC(kernel='rbf'))
svm_rbf.fit(train[feature_list], train['label'])
svm_predict = svm_rbf.predict(test[feature_list])
tend = time.time()

MetricsOfReliability(svm_rbf, test[feature_list], test['label'], svm_predict)
print('time: ' + str(tend - tstart))
```

[ ]:
```
#Measuring accuracy and compute time for the RF
tstart = time.time()
rf = RandomForestClassifier()
rf.fit(train[feature_list], train['label'])
rf_predict = rf.predict(test[feature_list])
tend = time.time()

MetricsOfReliability(rf, test[feature_list], test['label'], rf_predict)
print('time: ' + str(tend - tstart))
```