

best-3-1

September 21, 2021

```
[1]: base_transfer_set = ['01', '02', '04', '05', '08', '09', '12', '13', '16',  
    ↪ '17', '18', '20']  
target_transfer_set = ['03', '06', '07', '10', '11', '14', '15', '19']  
  
import random  
def random_combination(iterable, r):  
    "Random selection from itertools.combinations(iterable, r)"  
    pool = tuple(iterable)  
    n = len(pool)  
    indices = sorted(random.sample(range(n), r))  
    return tuple(pool[i] for i in indices)  
  
transfers_size_6 = []  
for i in range(4):  
    transfers_size_6.append(random_combination(target_transfer_set, 6))  
print(transfers_size_6)  
transfers_size_6 = [('03', '06', '07', '10', '11', '14'), ('03', '06', '07',  
    ↪ '10', '14', '15'), ('03', '06', '07', '10', '14', '15'), ('03', '07', '10',  
    ↪ '14', '15', '19')]  
for i, tmp in enumerate(transfers_size_6):  
    transfers_size_6[i] = list(transfers_size_6[i])  
print(transfers_size_6)  
  
transfers_size_4 = []  
for i in range(4):  
    transfers_size_4.append(random_combination(target_transfer_set, 4))  
print(transfers_size_4)  
transfers_size_4 = [('06', '10', '14', '15'), ('03', '10', '14', '19'), ('03',  
    ↪ '06', '10', '15'), ('03', '07', '10', '15')]  
for i, tmp in enumerate(transfers_size_4):  
    transfers_size_4[i] = list(transfers_size_4[i])  
print(transfers_size_4)  
  
transfers_size_3 = []  
for i in range(4):  
    transfers_size_3.append(random_combination(target_transfer_set, 3))  
print(transfers_size_3)
```

```

transfers_size_3 = [('07', '11', '14'), ('06', '07', '10'), ('03', '15', '19'),
↳ ('06', '14', '19')]
for i, tmp in enumerate(transfers_size_3):
    transfers_size_3[i] = list(transfers_size_3[i])
print(transfers_size_3)

transfers_size_2 = []
for i in range(4):
    transfers_size_2.append(random_combination(target_transfer_set, 2))
print(transfers_size_2)
transfers_size_2 = [('06', '10'), ('07', '11'), ('06', '15'), ('14', '15')]
for i, tmp in enumerate(transfers_size_2):
    transfers_size_2[i] = list(transfers_size_2[i])
print(transfers_size_2)

```

```

[('06', '07', '10', '11', '14', '15'), ('03', '07', '11', '14', '15', '19'),
('06', '07', '11', '14', '15', '19'), ('03', '10', '11', '14', '15', '19')]
[['03', '06', '07', '10', '11', '14'], ['03', '06', '07', '10', '14', '15'],
['03', '06', '07', '10', '14', '15'], ['03', '07', '10', '14', '15', '19']]
[('06', '10', '15', '19'), ('06', '07', '10', '15'), ('03', '07', '10', '19'),
('03', '10', '15', '19')]
[['06', '10', '14', '15'], ['03', '10', '14', '19'], ['03', '06', '10', '15'],
['03', '07', '10', '15']]
[('06', '07', '11'), ('03', '11', '19'), ('03', '11', '19'), ('06', '11', '15')]
[['07', '11', '14'], ['06', '07', '10'], ['03', '15', '19'], ['06', '14', '19']]
[('07', '11'), ('10', '11'), ('07', '10'), ('03', '15')]
[['06', '10'], ['07', '11'], ['06', '15'], ['14', '15']]

```

```

[2]: import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

```

```

def make_confusion_matrix(cf,
                           group_names=None,
                           categories='auto',
                           count=True,
                           percent=True,
                           cbar=True,
                           xyticks=True,
                           xyplotlabels=True,
                           sum_stats=True,
                           figsize=None,
                           cmap='Blues',
                           title=None):
    """

```

This function will make a pretty plot of an sklearn Confusion Matrix cm_
↳ using a Seaborn heatmap visualization.

Arguments

```
-----  
cf:                confusion matrix to be passed in  
group_names:       List of strings that represent the labels row by row to be  
→shown in each square.  
categories:        List of strings containing the categories to be displayed on  
→the x,y axis. Default is 'auto'  
count:             If True, show the raw number in the confusion matrix.  
→Default is True.  
normalize:         If True, show the proportions for each category. Default is  
→True.  
cbar:              If True, show the color bar. The cbar values are based off  
→the values in the confusion matrix.  
                   Default is True.  
xyticks:           If True, show x and y ticks. Default is True.  
xyplotlabels:      If True, show 'True Label' and 'Predicted Label' on the  
→figure. Default is True.  
sum_stats:         If True, display summary statistics below the figure.  
→Default is True.  
figsize:           Tuple representing the figure size. Default will be the  
→matplotlib rcParams value.  
cmap:              Colormap of the values displayed from matplotlib.pyplot.cm.  
→Default is 'Blues'  
                   See http://matplotlib.org/examples/color/colormaps\_reference.  
→html  
  
title:             Title for the heatmap. Default is None.  
'''  
  
# CODE TO GENERATE TEXT INSIDE EACH SQUARE  
blanks = ['' for i in range(cf.size)]  
  
if group_names and len(group_names)==cf.size:  
    group_labels = ["{}\n".format(value) for value in group_names]  
else:  
    group_labels = blanks  
  
if count:  
    group_counts = ["{0:0.0f}\n".format(value) for value in cf.flatten()]  
else:  
    group_counts = blanks  
  
if percent:  
    group_percentages = ["{0:.2%}".format(value) for value in cf.flatten()/  
→np.sum(cf)]
```

```

else:
    group_percentages = blanks

    box_labels = [f"{v1}-{v2}-{v3}".strip() for v1, v2, v3 in
→zip(group_labels,group_counts,group_percentages)]
    box_labels = np.asarray(box_labels).reshape(cf.shape[0],cf.shape[1])

# CODE TO GENERATE SUMMARY STATISTICS & TEXT FOR SUMMARY STATS
if sum_stats:
    #Accuracy is sum of diagonal divided by total observations
    accuracy = np.trace(cf) / float(np.sum(cf))

    #if it is a binary confusion matrix, show some more stats
    if len(cf)==2:
        #Metrics for Binary Confusion Matrices
        precision = cf[1,1] / sum(cf[:,1])
        recall    = cf[1,1] / sum(cf[1,:])
        f1_score  = 2*precision*recall / (precision + recall)
        stats_text = "\n\nAccuracy={:0.3f}\nPrecision={:0.3f}\nRecall={:0.
→3f}\nF1 Score={:0.3f}".format(
            accuracy,precision,recall,f1_score)
    else:
        stats_text = "\n\nAccuracy={:0.3f}".format(accuracy)
else:
    stats_text = ""

# SET FIGURE PARAMETERS ACCORDING TO OTHER ARGUMENTS
if figsize==None:
    #Get default figure size if not set
    figsize = plt.rcParams.get('figure.figsize')

if xyticks==False:
    #Do not show categories if xyticks is False
    categories=False

# MAKE THE HEATMAP VISUALIZATION
plt.figure(figsize=figsize)
sns.
→heatmap(cf,annot=box_labels,fmt="",cmap=cmap,cbar=cbar,xticklabels=categories,yticklabels=c

if xyplotlabels:
    plt.ylabel('True label')
    plt.xlabel('Predicted label' + stats_text)
else:

```

```
plt.xlabel(stats_text)

if title:
    plt.title(title)
```

```
-----
RuntimeError                                Traceback (most recent call last)
RuntimeError: module compiled against API version 0xe but this version of numpy
↳ is 0xd
```

```
-----
RuntimeError                                Traceback (most recent call last)
RuntimeError: module compiled against API version 0xe but this version of numpy
↳ is 0xd
```

```
[3]: import os
import pandas as pd
import warnings
warnings.filterwarnings("ignore")

def create_best_model(gesture_subset):
    gesture_subset.sort()
    print("Loading Dataset for gestures: ", gesture_subset)
    path = 'gestures-dataset'
    dataset = None

    samples = 0
    for subject in os.listdir(path):
        if os.path.isfile(os.path.join(path, subject)):
            continue
        if subject in ('U01', 'U02', 'U03', 'U04', 'U05', 'U06', 'U07', 'U08'):
            for gesture in os.listdir(os.path.join(path, subject)):
                if os.path.isfile(os.path.join(path, subject, gesture)):
                    continue
                gesture = str(gesture)
                if gesture not in gesture_subset:
                    continue
                for samplefile in os.listdir(os.path.join(path, subject,
↳gesture)):
                    if os.path.isfile(os.path.join(path, subject, gesture,
↳samplefile)):
                        df = pd.read_csv(os.path.join(path, subject, gesture,
↳samplefile), \
                                        sep = ' ', \
                                        names = ['System.currentTimeMillis()', \
```

```

        'System.nanoTime()', \
        'sample.timestamp', \
        'X', \
        'Y', \
        'Z' \
    ])
df = df[["sample.timestamp", "X", "Y", "Z"]]

start = df["sample.timestamp"][0]
df["sample.timestamp"] -= start
df["sample.timestamp"] /= 10000000
df["subject"] = subject
df["gesture"] = gesture
df["sample"] = str(samplefile[:-4])
samples += 1
#print(df)
if dataset is None:
    dataset = df.copy()
else:
    dataset = pd.concat([dataset, df])

dataset = dataset.sort_values(by=['gesture', 'subject', 'sample', 'sample.
→timestamp'])
data = dataset
print(str(samples) + " samples loaded")

print("Scaling Dataset for gestures: ", gesture_subset)
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
dataset_scaled = None

samples = 0
for i, gesture in enumerate(gesture_subset):
    df_gesture=data[data['gesture']==gesture]
    for j, subject in enumerate(df_gesture['subject'].unique()):
        df_subject=df_gesture[df_gesture['subject']==subject]
        for k, sample in enumerate(df_subject['sample'].unique()):
            df_sample=df_subject[df_subject['sample']==sample].copy()
            df_sample.sort_values(by=['sample.timestamp'])

            sc = scaler
            sc = sc.fit_transform(df_sample[["X", "Y", "Z"]])
            sc = pd.DataFrame(data=sc, columns=["X", "Y", "Z"])
            df_sample['X'] = sc['X']
            df_sample['Y'] = sc['Y']
            df_sample['Z'] = sc['Z']

```

```

        if dataset_scaled is None:
            dataset_scaled = df_sample.copy()
        else:
            dataset_scaled = pd.concat([dataset_scaled, df_sample])
        samples += 1
print(str(samples) + " samples scaled")
data = dataset_scaled

print("Cleaning Dataset for gestures: ", gesture_subset)
dataset_outliers = None
dataset_cleaned = None

samples = 0
outliers = 0
for i, gesture in enumerate(gesture_subset):
    df_gesture = data[data['gesture']==gesture]
    for j, subject in enumerate(df_gesture['subject'].unique()):
        df_subject = df_gesture[df_gesture['subject']==subject]

        time_mean = df_subject.groupby(["gesture", "subject", "sample"]).
→count().groupby(["gesture", "subject"]).agg({'sample.timestamp': ['mean']})
        time_std = df_subject.groupby(["gesture", "subject", "sample"]).
→count().groupby(["gesture", "subject"]).agg({'sample.timestamp': ['std']})
        time_max = time_mean['sample.timestamp'].iloc[0]['mean'] + 1.0 *
→time_std['sample.timestamp'].iloc[0]['std']
        time_min = time_mean['sample.timestamp'].iloc[0]['mean'] - 1.0 *
→time_std['sample.timestamp'].iloc[0]['std']
        for k, sample in enumerate(df_subject['sample'].unique()):
            df_sample=df_subject[df_subject['sample']==sample]
            df_sample_count = df_sample.count()['sample.timestamp']
            if df_sample_count < time_min or df_sample_count > time_max:
                if dataset_outliers is None:
                    dataset_outliers = df_sample.copy()
                else:
                    dataset_outliers = pd.concat([dataset_outliers,
→df_sample])
            outliers += 1
        else:
            if dataset_cleaned is None:
                dataset_cleaned = df_sample.copy()
            else:
                dataset_cleaned = pd.concat([dataset_cleaned,
→df_sample])
            samples += 1
print(str(samples) + " samples cleaned")
print(str(outliers) + " samples outliers")

```

```

data = dataset_cleaned

print("Time slicing Cleaned Dataset for gestures: ", gesture_subset)
dataset_timecut = None
samples = 0
damaged = 0
for i, gesture in enumerate(data['gesture'].unique()):
    df_gesture = data[data['gesture']==gesture]
    for j, subject in enumerate(df_gesture['subject'].unique()):
        df_subject = df_gesture[df_gesture['subject']==subject]
        time_max = 19 # 18 * 11 = 198
        for i, sample in enumerate(df_subject['sample'].unique()):
            df_sample = df_subject[df_subject['sample']==sample]
            df_sample_count = df_sample.count()['sample.timestamp']
            #print(df_sample_count)
            if df_sample_count >= time_max:
                df_sample = df_sample[df_sample['sample.timestamp'] <= (11_
↳* (time_max-1))]
                df_sample_count = df_sample.count()['sample.timestamp']
                #print(df_sample_count)
            elif df_sample_count < time_max:
                for tmp in range(df_sample_count * 11, (time_max) * 11, 11):
                    df = pd.DataFrame([[tmp, 0.0, 0.0, 0.0, gesture,
↳subject, sample]], columns=['sample.timestamp', 'X', 'Y', 'Z', 'gesture',
↳'subject', 'sample'])
                    df_sample = df_sample.append(df, ignore_index=True)
                    #print(df_sample)
                    df_sample_count = df_sample.count()['sample.timestamp']
                    #print(df_sample_count)
                    if df_sample_count != time_max:
                        damaged += 1
                        continue
                    if dataset_timecut is None:
                        dataset_timecut = df_sample.copy()
                    else:
                        dataset_timecut = pd.concat([dataset_timecut, df_sample])
                    samples += 1

dataset_cleaned = dataset_timecut
print(str(samples) + " cleaned samples sliced")
print(str(damaged) + " cleaned samples damaged")

data = dataset_outliers
print("Time slicing Outliers Dataset for gestures: ", gesture_subset)
dataset_timecut = None
samples = 0
damaged = 0

```



```

for i, gesture in enumerate(data['gesture'].unique()):
    df_gesture = data[data['gesture']==gesture]
    for j, subject in enumerate(df_gesture['subject'].unique()):
        df_subject = df_gesture[df_gesture['subject']==subject]
        time_max = 19 # 18 * 11 = 198
        for i, sample in enumerate(df_subject['sample'].unique()):
            df_sample = df_subject[df_subject['sample']==sample]
            df_sample_count = df_sample.count()['sample.timestamp']
            #print(df_sample_count)
            if df_sample_count >= time_max:
                df_sample = df_sample[df_sample['sample.timestamp'] <= (11
↳* (time_max-1))]
                df_sample_count = df_sample.count()['sample.timestamp']
                #print(df_sample_count)
            elif df_sample_count < time_max:
                for tmp in range(df_sample_count * 11, (time_max) * 11, 11):
                    df = pd.DataFrame([[tmp, 0.0, 0.0, 0.0, gesture,
↳subject, sample]], columns=['sample.timestamp', 'X', 'Y', 'Z', 'gesture',
↳'subject', 'sample'])
                    df_sample = df_sample.append(df, ignore_index=True)
                    #print(df_sample)
                    df_sample_count = df_sample.count()['sample.timestamp']
                    #print(df_sample_count)
                    if df_sample_count != time_max:
                        damaged += 1
                        continue
                    if dataset_timecut is None:
                        dataset_timecut = df_sample.copy()
                    else:
                        dataset_timecut = pd.concat([dataset_timecut, df_sample])
                    samples += 1

dataset_outliers = dataset_timecut
print(str(samples) + " outliers samples sliced")
print(str(damaged) + " outliers samples damaged")

data = dataset_cleaned

from keras.models import Sequential
from keras.layers import Bidirectional
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Dropout
from keras.optimizers import adam_v2
from keras.wrappers.scikit_learn import KerasClassifier
# from scikeras.wrappers import KerasClassifier
from sklearn.model_selection import StratifiedGroupKFold

```

```

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from keras.utils import np_utils
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
import numpy as np

# fix random seed for reproducibility
seed = 1000
np.random.seed(seed)
# create the dataset
def get_dataset(data):
    X_train = []
    Y_train = []
    groups = []
    for i, gesture in enumerate(data['gesture'].unique()):
        df_gesture = data[data['gesture']==gesture]
        for j, subject in enumerate(df_gesture['subject'].unique()):
            df_subject = df_gesture[df_gesture['subject']==subject]
            for k, sample in enumerate(df_subject['sample'].unique()):
                df_sample = df_subject[df_subject['sample']==sample]
                accel_vector = []
                for index, row in df_sample.sort_values(by='sample.
→timestamp').iterrows():
                    accel_vector.append([row['X'],row['Y'],row['Z']])
                accel_vector = np.asarray(accel_vector)
                X_train.append(accel_vector)
                Y_train.append(gesture)
                groups.append(subject)
    X_train = np.asarray(X_train)
    Y_train = LabelEncoder().fit_transform(Y_train)
    #print(Y_train)
    return X_train, Y_train, groups

# Function to create model, required for KerasClassifier
def create_model(dropout_rate=0.8, units=128, optimizer=adam_v2.
→Adam(learning_rate=0.001)):
    model = Sequential()
    model.add(
        Bidirectional(
            LSTM(
                units=units,
                input_shape=[19, 3]
            )
        )
    )
    model.add(Dropout(rate=dropout_rate))

```

```

        model.add(Dense(units=units, activation='relu'))
        model.add(Dense(len(gesture_subset), activation='softmax'))
        model.compile(loss='sparse_categorical_crossentropy',
→optimizer=optimizer, metrics=['accuracy'])
        #print(model.summary())
        return model

model = KerasClassifier(build_fn=create_model, verbose=0)
cv = StratifiedGroupKFold(n_splits=5, shuffle=True, random_state=1000)
# get the dataset
X, y, g = get_dataset(dataset_cleaned)
#cv = cv.split(X, y, g)
batch_size = [19]
epochs = [64, 128]
#epochs = [128]
units = [16, 32, 64, 128]
# units = [16]
dropout_rate = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
# dropout_rate = [0.5]
param_grid = dict(epochs=epochs, units=units, batch_size=batch_size,
→dropout_rate=dropout_rate)
print("Hyperparameter tuning started for Dataset for gestures: ",
→gesture_subset)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1,
→cv=cv, verbose=1)
grid_result = grid.fit(X, y, groups=g)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.
→best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
train_mean = grid_result.cv_results_['mean_fit_time']
train_std = grid_result.cv_results_['std_fit_time']
score_mean = grid_result.cv_results_['mean_score_time']
score_std = grid_result.cv_results_['std_score_time']
params = grid_result.cv_results_['params']
for mean, stdev, train_mean, train_std, score_mean, score_std, param in
→zip(means, stds, train_mean, train_std, score_mean, score_std, params):
    print("accuracy: %f (%f) train time: %f (%f) score time: %f (%f) with:
→%r" % (mean, stdev, train_mean, train_std, score_mean, score_std, param))
    print("Hyperparameter tuning completed for Dataset: ", gesture_subset)

model = grid_result.best_estimator_
import pickle

```

```

def save_model(model, gesture_subset):
    gesture_subset.sort()
    name = '-'.join(gesture_subset)
    # saving model
    pickle.dump(model.classes_, open(name + '_model_classes.pkl', 'wb'))
    model.model.save(name + '_lstm')
    print("Saving model to disk started for Dataset gestures: ", gesture_subset)
    save_model(model, gesture_subset)
    print("Saving model to disk completed for Dataset gestures: ",
    ↳gesture_subset)

import tensorflow as tf
def load_model(gesture_subset):
    gesture_subset.sort()
    name = '-'.join(gesture_subset)
    # loading model
    build_model = lambda: tf.keras.models.load_model(name + '_lstm')
    classifier = KerasClassifier(build_fn=build_model, epochs=1,
    ↳batch_size=10, verbose=0)
    classifier.classes_ = pickle.load(open(name + '_model_classes.
    ↳pkl', 'rb'))
    classifier.model = build_model()
    return classifier
    print("Loading model to disk started for Dataset gestures: ",
    ↳gesture_subset)
    model = load_model(gesture_subset)
    #print(model.model.sumint("Loading model to disk completed for Dataset
    ↳gestures: ", gesture_subset)

    print("Testing model against outliers for Dataset gestures: ",
    ↳gesture_subset)
    data = dataset_outliers
    X, y, g = get_dataset(dataset_outliers)
    y_pred = model.predict(X)

    from sklearn.metrics import classification_report
    print(classification_report(y, y_pred, target_names=gesture_subset))

    from sklearn.metrics import confusion_matrix
    cf_matrix = confusion_matrix(y, y_pred)
    make_confusion_matrix(cf_matrix, categories=gesture_subset, figsize=[8,8])
    return grid_result
base_transfer_set = ['01', '02', '04', '05', '08', '09', '12', '13', '16',
    ↳'17', '18', '20']
dataset = transfers_size_3[1]

```

```
results = create_best_model(dataset)
```

```
Loading Dataset for gestures: ['06', '07', '10']
489 samples loaded
Scaling Dataset for gestures: ['06', '07', '10']
489 samples scaled
Cleaning Dataset for gestures: ['06', '07', '10']
365 samples cleaned
124 samples outliers
Time slicing Cleaned Dataset for gestures: ['06', '07', '10']
364 cleaned samples sliced
1 cleaned samples damaged
Time slicing Outliers Dataset for gestures: ['06', '07', '10']
124 outliers samples sliced
0 outliers samples damaged

2021-09-21 12:19:51.162380: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libcudart.so.11.0'; dLError: libcudart.so.11.0: cannot open
shared object file: No such file or directory
2021-09-21 12:19:51.162418: I tensorflow/stream_executor/cuda/cudart_stub.cc:29]
Ignore above cudart dLError if you do not have a GPU set up on your machine.

Hyperparameter tuning started for Dataset for gestures: ['06', '07', '10']
Fitting 5 folds for each of 72 candidates, totalling 360 fits

2021-09-21 12:19:55.137476: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libcuda.so.1'; dLError: libcuda.so.1: cannot open shared object
file: No such file or directory
2021-09-21 12:19:55.137513: W
tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to cuInit:
UNKNOWN ERROR (303)
2021-09-21 12:19:55.137533: I
tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not
appear to be running on this host (mqx-public): /proc/driver/nvidia/version does
not exist
2021-09-21 12:19:55.138227: I tensorflow/core/platform/cpu_feature_guard.cc:142]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations: AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-09-21 12:19:55.316112: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR
Optimization Passes are enabled (registered 2)

Best: 0.995652 using {'batch_size': 19, 'dropout_rate': 0.2, 'epochs': 64,
'units': 16}
```

accuracy: 0.958597 (0.077250) train time: 28.727976 (2.164540) score time:
 2.514681 (0.304153) with: {'batch_size': 19, 'dropout_rate': 0.1, 'epochs': 64,
 'units': 16}
 accuracy: 0.947628 (0.093793) train time: 29.624809 (1.430083) score time:
 2.516717 (0.632711) with: {'batch_size': 19, 'dropout_rate': 0.1, 'epochs': 64,
 'units': 32}
 accuracy: 0.958597 (0.077250) train time: 42.125547 (3.147353) score time:
 4.292558 (0.790308) with: {'batch_size': 19, 'dropout_rate': 0.1, 'epochs': 64,
 'units': 64}
 accuracy: 0.944998 (0.082278) train time: 64.207527 (5.599965) score time:
 7.429446 (1.647805) with: {'batch_size': 19, 'dropout_rate': 0.1, 'epochs': 64,
 'units': 128}
 accuracy: 0.984684 (0.025336) train time: 58.278774 (3.548766) score time:
 4.885319 (1.331765) with: {'batch_size': 19, 'dropout_rate': 0.1, 'epochs': 128,
 'units': 16}
 accuracy: 0.951976 (0.085140) train time: 58.212655 (2.565089) score time:
 5.076703 (0.125850) with: {'batch_size': 19, 'dropout_rate': 0.1, 'epochs': 128,
 'units': 32}
 accuracy: 0.933794 (0.087472) train time: 68.093532 (4.581311) score time:
 4.858225 (1.213640) with: {'batch_size': 19, 'dropout_rate': 0.1, 'epochs': 128,
 'units': 64}
 accuracy: 0.934709 (0.076146) train time: 114.891301 (10.070906) score time:
 7.313478 (1.379688) with: {'batch_size': 19, 'dropout_rate': 0.1, 'epochs': 128,
 'units': 128}
 accuracy: 0.995652 (0.008696) train time: 48.283257 (10.366000) score time:
 4.909572 (0.742977) with: {'batch_size': 19, 'dropout_rate': 0.2, 'epochs': 64,
 'units': 16}
 accuracy: 0.960870 (0.078261) train time: 41.850421 (3.497174) score time:
 3.725575 (0.870143) with: {'batch_size': 19, 'dropout_rate': 0.2, 'epochs': 64,
 'units': 32}
 accuracy: 0.965217 (0.069565) train time: 40.904355 (2.826110) score time:
 3.372291 (0.288945) with: {'batch_size': 19, 'dropout_rate': 0.2, 'epochs': 64,
 'units': 64}
 accuracy: 0.938217 (0.090364) train time: 59.724277 (5.154467) score time:
 3.805203 (0.513901) with: {'batch_size': 19, 'dropout_rate': 0.2, 'epochs': 64,
 'units': 128}
 accuracy: 0.965217 (0.069565) train time: 62.079621 (3.872444) score time:
 4.468459 (1.216072) with: {'batch_size': 19, 'dropout_rate': 0.2, 'epochs': 128,
 'units': 16}
 accuracy: 0.973913 (0.052174) train time: 62.789735 (2.554934) score time:
 4.262609 (0.581523) with: {'batch_size': 19, 'dropout_rate': 0.2, 'epochs': 128,
 'units': 32}
 accuracy: 0.956324 (0.076496) train time: 66.218177 (3.113607) score time:
 3.475106 (0.691621) with: {'batch_size': 19, 'dropout_rate': 0.2, 'epochs': 128,
 'units': 64}
 accuracy: 0.949185 (0.074448) train time: 103.500090 (6.728027) score time:
 6.304807 (0.641817) with: {'batch_size': 19, 'dropout_rate': 0.2, 'epochs': 128,
 'units': 128}

accuracy: 0.969368 (0.050672) train time: 45.228130 (3.347242) score time:
 5.212441 (1.561165) with: {'batch_size': 19, 'dropout_rate': 0.3, 'epochs': 64,
 'units': 16}
 accuracy: 0.989032 (0.016840) train time: 42.608869 (6.067965) score time:
 6.272259 (1.275495) with: {'batch_size': 19, 'dropout_rate': 0.3, 'epochs': 64,
 'units': 32}
 accuracy: 0.953693 (0.065177) train time: 44.708421 (3.810787) score time:
 6.996639 (1.211184) with: {'batch_size': 19, 'dropout_rate': 0.3, 'epochs': 64,
 'units': 64}
 accuracy: 0.964581 (0.049225) train time: 54.977149 (5.707001) score time:
 3.645342 (0.947717) with: {'batch_size': 19, 'dropout_rate': 0.3, 'epochs': 64,
 'units': 128}
 accuracy: 0.958399 (0.067461) train time: 55.672750 (1.509383) score time:
 4.437699 (1.819855) with: {'batch_size': 19, 'dropout_rate': 0.3, 'epochs': 128,
 'units': 16}
 accuracy: 0.969565 (0.060870) train time: 58.736584 (4.957239) score time:
 5.709986 (1.665411) with: {'batch_size': 19, 'dropout_rate': 0.3, 'epochs': 128,
 'units': 32}
 accuracy: 0.956522 (0.086957) train time: 64.181018 (3.504604) score time:
 5.495872 (0.574491) with: {'batch_size': 19, 'dropout_rate': 0.3, 'epochs': 128,
 'units': 64}
 accuracy: 0.962309 (0.048316) train time: 103.166957 (10.264704) score time:
 4.784381 (1.016534) with: {'batch_size': 19, 'dropout_rate': 0.3, 'epochs': 128,
 'units': 128}
 accuracy: 0.986759 (0.017494) train time: 42.651169 (3.130078) score time:
 4.226280 (1.244605) with: {'batch_size': 19, 'dropout_rate': 0.4, 'epochs': 64,
 'units': 16}
 accuracy: 0.980256 (0.033914) train time: 43.551614 (3.593423) score time:
 3.777581 (1.241243) with: {'batch_size': 19, 'dropout_rate': 0.4, 'epochs': 64,
 'units': 32}
 accuracy: 0.955688 (0.056120) train time: 47.884863 (5.949116) score time:
 4.789970 (0.309974) with: {'batch_size': 19, 'dropout_rate': 0.4, 'epochs': 64,
 'units': 64}
 accuracy: 0.981973 (0.018112) train time: 64.648214 (3.603187) score time:
 5.869445 (1.277078) with: {'batch_size': 19, 'dropout_rate': 0.4, 'epochs': 64,
 'units': 128}
 accuracy: 0.991304 (0.017391) train time: 54.717876 (4.036916) score time:
 5.293723 (0.655121) with: {'batch_size': 19, 'dropout_rate': 0.4, 'epochs': 128,
 'units': 16}
 accuracy: 0.949543 (0.083919) train time: 52.404223 (4.212698) score time:
 5.652757 (1.022799) with: {'batch_size': 19, 'dropout_rate': 0.4, 'epochs': 128,
 'units': 32}
 accuracy: 0.969565 (0.060870) train time: 64.389011 (2.269035) score time:
 4.857775 (1.160703) with: {'batch_size': 19, 'dropout_rate': 0.4, 'epochs': 128,
 'units': 64}
 accuracy: 0.955929 (0.060157) train time: 100.902816 (6.674400) score time:
 6.573549 (0.933046) with: {'batch_size': 19, 'dropout_rate': 0.4, 'epochs': 128,
 'units': 128}

accuracy: 0.986957 (0.026087) train time: 42.874212 (6.444487) score time:
 6.037098 (1.525870) with: {'batch_size': 19, 'dropout_rate': 0.5, 'epochs': 64,
 'units': 16}
 accuracy: 0.967292 (0.059895) train time: 37.946387 (3.468552) score time:
 4.250661 (0.769870) with: {'batch_size': 19, 'dropout_rate': 0.5, 'epochs': 64,
 'units': 32}
 accuracy: 0.947826 (0.104348) train time: 42.694838 (2.785674) score time:
 5.143862 (1.007027) with: {'batch_size': 19, 'dropout_rate': 0.5, 'epochs': 64,
 'units': 64}
 accuracy: 0.977187 (0.029226) train time: 65.930600 (5.749390) score time:
 5.354228 (1.655168) with: {'batch_size': 19, 'dropout_rate': 0.5, 'epochs': 64,
 'units': 128}
 accuracy: 0.965217 (0.069565) train time: 63.142544 (4.631252) score time:
 6.268314 (1.155303) with: {'batch_size': 19, 'dropout_rate': 0.5, 'epochs': 128,
 'units': 16}
 accuracy: 0.982609 (0.034783) train time: 56.531056 (6.260335) score time:
 5.194353 (1.082179) with: {'batch_size': 19, 'dropout_rate': 0.5, 'epochs': 128,
 'units': 32}
 accuracy: 0.975988 (0.042570) train time: 64.268277 (4.479567) score time:
 3.755100 (1.153742) with: {'batch_size': 19, 'dropout_rate': 0.5, 'epochs': 128,
 'units': 64}
 accuracy: 0.962106 (0.038696) train time: 100.665218 (12.080553) score time:
 5.259629 (0.515972) with: {'batch_size': 19, 'dropout_rate': 0.5, 'epochs': 128,
 'units': 128}
 accuracy: 0.960870 (0.078261) train time: 45.996486 (8.715079) score time:
 7.089840 (1.578425) with: {'batch_size': 19, 'dropout_rate': 0.6, 'epochs': 64,
 'units': 16}
 accuracy: 0.969565 (0.060870) train time: 42.048390 (4.365861) score time:
 4.129057 (0.544823) with: {'batch_size': 19, 'dropout_rate': 0.6, 'epochs': 64,
 'units': 32}
 accuracy: 0.973635 (0.041489) train time: 42.564718 (3.539368) score time:
 5.001713 (1.152548) with: {'batch_size': 19, 'dropout_rate': 0.6, 'epochs': 64,
 'units': 64}
 accuracy: 0.973555 (0.042123) train time: 62.259833 (6.895728) score time:
 5.675992 (0.388936) with: {'batch_size': 19, 'dropout_rate': 0.6, 'epochs': 64,
 'units': 128}
 accuracy: 0.991948 (0.009914) train time: 60.650330 (4.239510) score time:
 5.874052 (0.807463) with: {'batch_size': 19, 'dropout_rate': 0.6, 'epochs': 128,
 'units': 16}
 accuracy: 0.960870 (0.078261) train time: 62.344010 (3.277863) score time:
 5.554357 (0.941207) with: {'batch_size': 19, 'dropout_rate': 0.6, 'epochs': 128,
 'units': 32}
 accuracy: 0.976632 (0.032568) train time: 65.875722 (9.053256) score time:
 5.164982 (1.595513) with: {'batch_size': 19, 'dropout_rate': 0.6, 'epochs': 128,
 'units': 64}
 accuracy: 0.946159 (0.049296) train time: 96.715356 (12.499681) score time:
 6.100977 (2.003792) with: {'batch_size': 19, 'dropout_rate': 0.6, 'epochs': 128,
 'units': 128}

accuracy: 0.966854 (0.050529) train time: 46.203488 (4.927553) score time:
 5.971162 (1.178990) with: {'batch_size': 19, 'dropout_rate': 0.7, 'epochs': 64,
 'units': 16}
 accuracy: 0.967292 (0.059895) train time: 42.439468 (5.604760) score time:
 5.488648 (1.638237) with: {'batch_size': 19, 'dropout_rate': 0.7, 'epochs': 64,
 'units': 32}
 accuracy: 0.969565 (0.060870) train time: 43.543643 (3.753062) score time:
 5.010028 (1.453537) with: {'batch_size': 19, 'dropout_rate': 0.7, 'epochs': 64,
 'units': 64}
 accuracy: 0.962229 (0.049444) train time: 59.296703 (4.874452) score time:
 5.830198 (1.997741) with: {'batch_size': 19, 'dropout_rate': 0.7, 'epochs': 64,
 'units': 128}
 accuracy: 0.967095 (0.050521) train time: 58.555804 (6.405615) score time:
 7.034611 (0.858613) with: {'batch_size': 19, 'dropout_rate': 0.7, 'epochs': 128,
 'units': 16}
 accuracy: 0.969207 (0.050648) train time: 62.177167 (4.479775) score time:
 6.208620 (1.622969) with: {'batch_size': 19, 'dropout_rate': 0.7, 'epochs': 128,
 'units': 32}
 accuracy: 0.960870 (0.078261) train time: 66.965386 (4.258749) score time:
 6.107749 (1.530742) with: {'batch_size': 19, 'dropout_rate': 0.7, 'epochs': 128,
 'units': 64}
 accuracy: 0.975865 (0.032516) train time: 102.487535 (6.068823) score time:
 5.244518 (1.221410) with: {'batch_size': 19, 'dropout_rate': 0.7, 'epochs': 128,
 'units': 128}
 accuracy: 0.958474 (0.066725) train time: 42.853420 (3.474613) score time:
 5.537899 (1.605856) with: {'batch_size': 19, 'dropout_rate': 0.8, 'epochs': 64,
 'units': 16}
 accuracy: 0.960870 (0.078261) train time: 41.775511 (4.093496) score time:
 4.279222 (1.193463) with: {'batch_size': 19, 'dropout_rate': 0.8, 'epochs': 64,
 'units': 32}
 accuracy: 0.967292 (0.059895) train time: 42.703389 (2.069338) score time:
 4.427480 (0.441213) with: {'batch_size': 19, 'dropout_rate': 0.8, 'epochs': 64,
 'units': 64}
 accuracy: 0.988593 (0.014613) train time: 58.929623 (2.743325) score time:
 5.311649 (1.142150) with: {'batch_size': 19, 'dropout_rate': 0.8, 'epochs': 64,
 'units': 128}
 accuracy: 0.978261 (0.043478) train time: 58.077608 (1.632241) score time:
 4.425833 (0.684692) with: {'batch_size': 19, 'dropout_rate': 0.8, 'epochs': 128,
 'units': 16}
 accuracy: 0.984246 (0.019468) train time: 61.042786 (2.631462) score time:
 4.848665 (0.765218) with: {'batch_size': 19, 'dropout_rate': 0.8, 'epochs': 128,
 'units': 32}
 accuracy: 0.971202 (0.042223) train time: 65.850206 (1.703957) score time:
 6.176863 (0.907364) with: {'batch_size': 19, 'dropout_rate': 0.8, 'epochs': 128,
 'units': 64}
 accuracy: 0.988951 (0.016843) train time: 101.640610 (8.530683) score time:
 5.529232 (1.627288) with: {'batch_size': 19, 'dropout_rate': 0.8, 'epochs': 128,
 'units': 128}

```

accuracy: 0.962375 (0.053584) train time: 46.856775 (8.661397) score time:
5.101081 (1.245911) with: {'batch_size': 19, 'dropout_rate': 0.9, 'epochs': 64,
'units': 16}
accuracy: 0.982411 (0.025389) train time: 43.235169 (6.311050) score time:
3.988344 (0.890168) with: {'batch_size': 19, 'dropout_rate': 0.9, 'epochs': 64,
'units': 32}
accuracy: 0.965503 (0.048908) train time: 44.578580 (1.830546) score time:
4.725015 (1.202077) with: {'batch_size': 19, 'dropout_rate': 0.9, 'epochs': 64,
'units': 64}
accuracy: 0.963874 (0.041587) train time: 56.411203 (4.984472) score time:
4.534045 (1.054011) with: {'batch_size': 19, 'dropout_rate': 0.9, 'epochs': 64,
'units': 128}
accuracy: 0.976552 (0.032538) train time: 63.671777 (5.377779) score time:
5.716479 (1.550156) with: {'batch_size': 19, 'dropout_rate': 0.9, 'epochs': 128,
'units': 16}
accuracy: 0.979663 (0.025266) train time: 62.000082 (3.629235) score time:
6.158600 (0.646127) with: {'batch_size': 19, 'dropout_rate': 0.9, 'epochs': 128,
'units': 32}
accuracy: 0.958801 (0.050286) train time: 83.391822 (12.464714) score time:
5.881160 (2.185166) with: {'batch_size': 19, 'dropout_rate': 0.9, 'epochs': 128,
'units': 64}
accuracy: 0.986957 (0.026087) train time: 109.247177 (5.923704) score time:
7.722963 (1.514510) with: {'batch_size': 19, 'dropout_rate': 0.9, 'epochs': 128,
'units': 128}
Hyperparameter tuning completed for Dataset: ['06', '07', '10']
Saving model to disk started for Dataset gestures: ['06', '07', '10']

2021-09-21 18:51:44.820510: W tensorflow/python/util/util.cc:348] Sets are not
currently considered sequences, but this may change in the future, so consider
avoiding using them.
WARNING:absl:Found untraced functions such as lstm_cell_1081_layer_call_fn,
lstm_cell_1081_layer_call_and_return_conditional_losses,
lstm_cell_1082_layer_call_fn,
lstm_cell_1082_layer_call_and_return_conditional_losses,
lstm_cell_1081_layer_call_fn while saving (showing 5 of 10). These functions
will not be directly callable after loading.

INFO:tensorflow:Assets written to: 06-07-10_lstm/assets

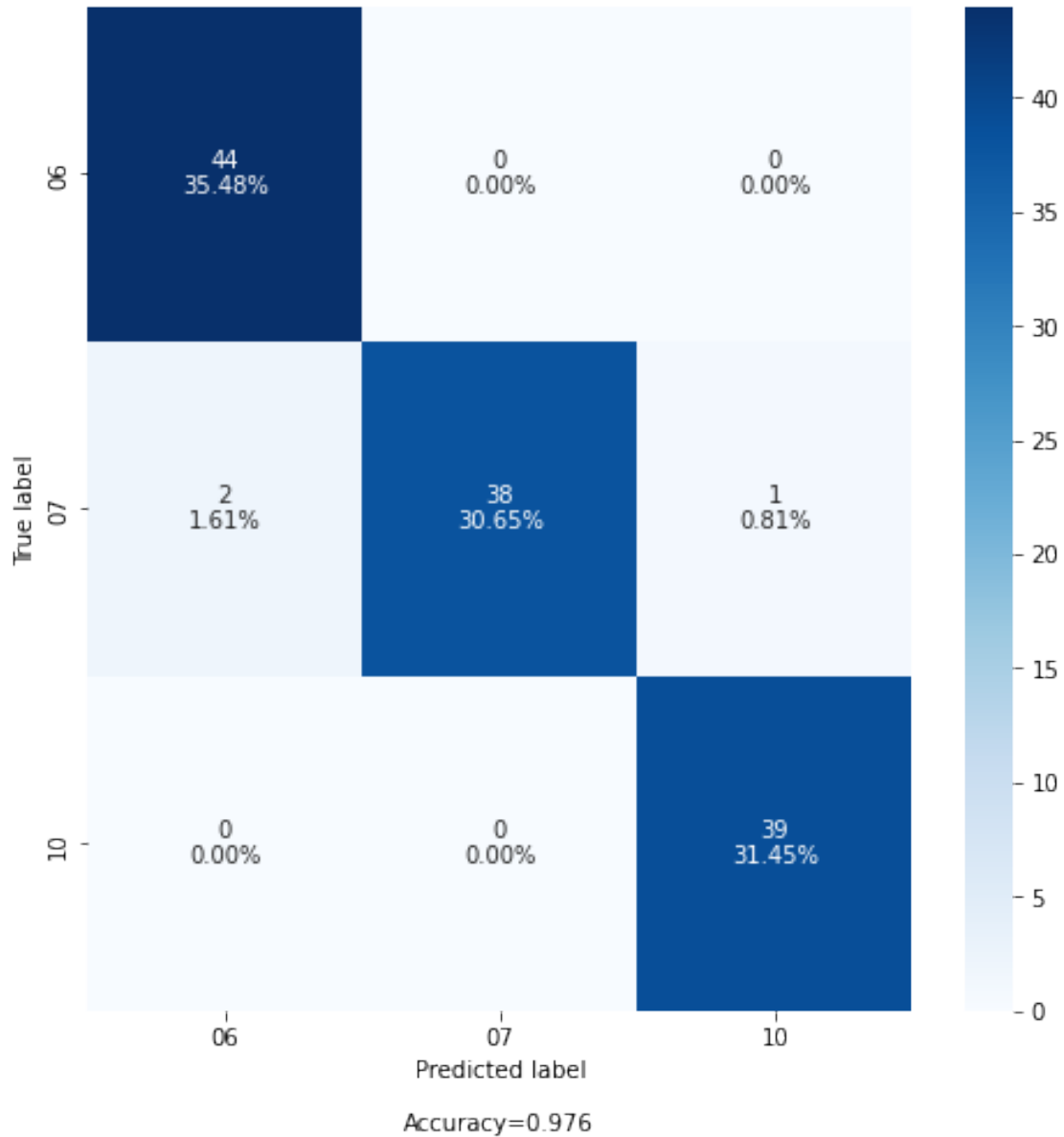
INFO:tensorflow:Assets written to: 06-07-10_lstm/assets

Saving model to disk completed for Dataset gestures: ['06', '07', '10']
Loading model to disk started for Dataset gestures: ['06', '07', '10']
Testing model against outliers for Dataset gestures: ['06', '07', '10']
      precision    recall  f1-score   support

     06         0.96         1.00         0.98         44
     07         1.00         0.93         0.96         41
     10         0.97         1.00         0.99         39

```

accuracy			0.98	124
macro avg	0.98	0.98	0.98	124
weighted avg	0.98	0.98	0.98	124



```
[4]: import os
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```

```

baseset = dataset

def evaluate_model(baseset):
    print("Baseset: ", baseset)
    print("Loadind Dataset: ", baseset)
    path = 'gestures-dataset'
    dataset = None

    samples = 0
    for subject in os.listdir(path):
        if os.path.isfile(os.path.join(path, subject)):
            continue
        if subject in ('U01', 'U02', 'U03', 'U04', 'U05', 'U06', 'U07', 'U08'):
            for gesture in os.listdir(os.path.join(path, subject)):
                if os.path.isfile(os.path.join(path, subject, gesture)):
                    continue
                gesture = str(gesture)
                if gesture not in baseset:
                    continue
                for samplefile in os.listdir(os.path.join(path, subject,
↪gesture)):
                    if os.path.isfile(os.path.join(path, subject, gesture,
↪samplefile)):
                        df = pd.read_csv(os.path.join(path, subject, gesture,
↪samplefile), \
                                sep = ' ', \
                                names = ['System.currentTimeMillis()', \
                                        'System.nanoTime()', \
                                        'sample.timestamp', \
                                        'X', \
                                        'Y', \
                                        'Z' \
                                        ])
                        df = df[["sample.timestamp", "X", "Y", "Z"]]

                        start = df["sample.timestamp"][0]
                        df["sample.timestamp"] -= start
                        df["sample.timestamp"] /= 10000000
                        df["subject"] = subject
                        df["gesture"] = gesture
                        df["sample"] = str(samplefile[:-4])
                        samples += 1
                        #print(df)
                        if dataset is None:
                            dataset = df.copy()
                        else:
                            dataset = pd.concat([dataset, df])

```

```

dataset = dataset.sort_values(by=['gesture', 'subject', 'sample', 'sample.
→timestamp'])
data = dataset
print(str(samples) + " samples loaded")

print("Scaling Dataset: ", baseset)
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
dataset_scaled = None

samples = 0
for i, gesture in enumerate(baseset):
    df_gesture=data[data['gesture']==gesture]
    for j, subject in enumerate(df_gesture['subject'].unique()):
        df_subject=df_gesture[df_gesture['subject']==subject]
        for k, sample in enumerate(df_subject['sample'].unique()):
            df_sample=df_subject[df_subject['sample']==sample].copy()
            df_sample.sort_values(by=['sample.timestamp'])

            sc = scaler
            sc = sc.fit_transform(df_sample[["X", "Y", "Z"]])
            sc = pd.DataFrame(data=sc, columns=["X", "Y", "Z"])
            df_sample['X'] = sc['X']
            df_sample['Y'] = sc['Y']
            df_sample['Z'] = sc['Z']
            if dataset_scaled is None:
                dataset_scaled = df_sample.copy()
            else:
                dataset_scaled = pd.concat([dataset_scaled, df_sample])
            samples += 1
print(str(samples) + " samples scaled")
data = dataset_scaled

print("Cleaning Dataset: ", baseset)
dataset_outliers = None
dataset_cleaned = None

samples = 0
outliers = 0
for i, gesture in enumerate(baseset):
    df_gesture = data[data['gesture']==gesture]
    for j, subject in enumerate(df_gesture['subject'].unique()):
        df_subject = df_gesture[df_gesture['subject']==subject]

```

```

        time_mean = df_subject.groupby(["gesture", "subject", "sample"]).
→count().groupby(["gesture", "subject"]).agg({'sample.timestamp': ['mean']})
        time_std = df_subject.groupby(["gesture", "subject", "sample"]).
→count().groupby(["gesture", "subject"]).agg({'sample.timestamp': ['std']})
        time_max = time_mean['sample.timestamp'].iloc[0]['mean'] + 1.0 *
→time_std['sample.timestamp'].iloc[0]['std']
        time_min = time_mean['sample.timestamp'].iloc[0]['mean'] - 1.0 *
→time_std['sample.timestamp'].iloc[0]['std']
        for k, sample in enumerate(df_subject['sample'].unique()):
            df_sample=df_subject[df_subject['sample']==sample]
            df_sample_count = df_sample.count()['sample.timestamp']
            if df_sample_count < time_min or df_sample_count > time_max:
                if dataset_outliers is None:
                    dataset_outliers = df_sample.copy()
                else:
                    dataset_outliers = pd.concat([dataset_outliers,
→df_sample])
                    outliers += 1
            else:
                if dataset_cleaned is None:
                    dataset_cleaned = df_sample.copy()
                else:
                    dataset_cleaned = pd.concat([dataset_cleaned,
→df_sample])
                    samples += 1
        print(str(samples) + " samples cleaned")
        print(str(outliers) + " samples outliers")
        data = dataset_cleaned

        print("Time slicing Cleaned Dataset: ", baseset)
        dataset_timecut = None
        samples = 0
        damaged = 0
        for i, gesture in enumerate(data['gesture'].unique()):
            df_gesture = data[data['gesture']==gesture]
            for j, subject in enumerate(df_gesture['subject'].unique()):
                df_subject = df_gesture[df_gesture['subject']==subject]
                time_max = 19 # 18 * 11 = 198
                for i, sample in enumerate(df_subject['sample'].unique()):
                    df_sample = df_subject[df_subject['sample']==sample]
                    df_sample_count = df_sample.count()['sample.timestamp']
                    #print(df_sample_count)
                    if df_sample_count >= time_max:
                        df_sample = df_sample[df_sample['sample.timestamp'] <= (11
→* (time_max-1))]
                        df_sample_count = df_sample.count()['sample.timestamp']

```

```

        #print(df_sample_count)
    elif df_sample_count < time_max:
        for tmp in range(df_sample_count * 11, (time_max) * 11, 11):
            df = pd.DataFrame([[tmp, 0.0, 0.0, 0.0, gesture,
→subject, sample]], columns=['sample.timestamp', 'X', 'Y', 'Z', 'gesture',
→'subject', 'sample'])

            df_sample = df_sample.append(df, ignore_index=True)
        #print(df_sample)
        df_sample_count = df_sample.count()['sample.timestamp']
        #print(df_sample_count)
        if df_sample_count != time_max:
            damaged += 1
            continue
        if dataset_timecut is None:
            dataset_timecut = df_sample.copy()
        else:
            dataset_timecut = pd.concat([dataset_timecut, df_sample])
        samples += 1

dataset_cleaned = dataset_timecut
print(str(samples) + " cleaned samples sliced")
print(str(damaged) + " cleaned samples damaged")

data = dataset_outliers
print("Time slicing Outliers Dataset: ", baseset)
dataset_timecut = None
samples = 0
damaged = 0
for i, gesture in enumerate(data['gesture'].unique()):
    df_gesture = data[data['gesture']==gesture]
    for j, subject in enumerate(df_gesture['subject'].unique()):
        df_subject = df_gesture[df_gesture['subject']==subject]
        time_max = 19 # 18 * 11 = 198
        for i, sample in enumerate(df_subject['sample'].unique()):
            df_sample = df_subject[df_subject['sample']==sample]
            df_sample_count = df_sample.count()['sample.timestamp']
            #print(df_sample_count)
            if df_sample_count >= time_max:
                df_sample = df_sample[df_sample['sample.timestamp'] <= (11
→* (time_max-1))]
                df_sample_count = df_sample.count()['sample.timestamp']
                #print(df_sample_count)
            elif df_sample_count < time_max:
                for tmp in range(df_sample_count * 11, (time_max) * 11, 11):
                    df = pd.DataFrame([[tmp, 0.0, 0.0, 0.0, gesture,
→subject, sample]], columns=['sample.timestamp', 'X', 'Y', 'Z', 'gesture',
→'subject', 'sample'])

```

```

        df_sample = df_sample.append(df, ignore_index=True)
        #print(df_sample)
        df_sample_count = df_sample.count()['sample.timestamp']
        #print(df_sample_count)
        if df_sample_count != time_max:
            damaged += 1
            continue
        if dataset_timecut is None:
            dataset_timecut = df_sample.copy()
        else:
            dataset_timecut = pd.concat([dataset_timecut, df_sample])
        samples += 1

dataset_outliers = dataset_timecut
print(str(samples) + " outliers samples sliced")
print(str(damaged) + " outliers samples damaged")

from keras import backend as K
data = dataset_cleaned
from keras.models import Sequential
from keras.layers import Bidirectional
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Dropout
from keras.optimizers import adam_v2
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import StratifiedGroupKFold
from sklearn.model_selection import cross_validate
from sklearn.model_selection import GridSearchCV
from keras.utils import np_utils
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
import numpy as np
import tensorflow as tf

# fix random seed for reproducibility
seed = 1000
np.random.seed(seed)
# create the dataset
def get_dataset(data, index=[]):
    X_train = []
    Y_train = []
    groups = []
    samples_idx=0
    for i, gesture in enumerate(data['gesture'].unique()):
        df_gesture = data[data['gesture']==gesture]

```



```

        for j, subject in enumerate(df_gesture['subject'].unique()):
            df_subject = df_gesture[df_gesture['subject']==subject]
            for k, sample in enumerate(df_subject['sample'].unique()):
                df_sample = df_subject[df_subject['sample']==sample]
                accel_vector = []
                for idx, row in df_sample.sort_values(by='sample.
→timestamp').iterrows():
                    accel_vector.append([row['X'],row['Y'],row['Z']])
                accel_vector = np.asarray(accel_vector)
                if len(index)==0:
                    X_train.append(accel_vector)
                    Y_train.append(gesture)
                    groups.append(subject)
                else:
                    if samples_idx in index:
                        X_train.append(accel_vector)
                        Y_train.append(gesture)
                        groups.append(subject)
                    samples_idx+=1
X_train = np.asarray(X_train)
Y_train = LabelEncoder().fit_transform(Y_train)
#print(Y_train)
return X_train, Y_train, groups

def build_model(baseset):
    baseset.sort()
    basename = '-'.join(baseset)
    basemodel = tf.keras.models.load_model(basename + '_lstm')
    basemodel.build([None, 19, 3])
    #print(model.summary())
    basemodel.compile(loss='sparse_categorical_crossentropy',
→optimizer=adam_v2.Adam(learning_rate=0.001), metrics=['accuracy'])
    return basemodel

# Function to create model, required for KerasClassifier
import pickle
def load_classifier(baseset):
    baseset.sort()
    basename = '-'.join(baseset)
    classifier = KerasClassifier(build_fn=build_model, baseset=baseset,
→epochs=64, batch_size=19, verbose=0)
    classifier.classes_ = pickle.load(open(basename + '_model_classes.
→pkl','rb'))
    classifier.model = build_model(baseset)
    return classifier

```

```

# print(model.model.summary())
# print(model.classes_)
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

for n_splits in [5]:
    for epoch in [[results.best_params_['epochs']]]:
        cv = StratifiedGroupKFold(n_splits=n_splits, shuffle=True,
→ random_state=(1000+epoch[0]))
        X, y, g = get_dataset(dataset_cleaned)

        # Initialize the accuracy of the models to blank list. The accuracy
→ of each model will be appended to this list
        accuracy_model = []
        best_estimator = None
        # Initialize the array to zero which will store the confusion matrix
        array = None
        outliers = None

        report_cleaned = None
        report_outliers = None

        print("Processing started for split estimator: " + str(n_splits) +
→ ", epochs: " + str(epoch))
        # Iterate over each train-test split
        fold = 1
        for train_index, test_index in cv.split(X, y, g):
            # print(test_index)
            if len(test_index) == 0 or len(train_index) == 0:
                continue
            print("Processing ", fold, "-fold")
            fold += 1

            classifier = load_classifier(basetest)
            # Split train-test (Inverted)
            X_train, y_train, group_train = get_dataset(dataset_cleaned,
→ train_index)
            X_test, y_test, group_test = get_dataset(dataset_cleaned,
→ test_index)
            X_outliers, y_outliers, group_test =
→ get_dataset(dataset_outliers)
            # Train the model
            History = classifier.fit(X_train, y_train, epochs=epoch[0])
            # Append to accuracy_model the accuracy of the model
            accuracy_model.append(accuracy_score(y_test, classifier.
→ predict(X_test), normalize=True))

```

```

        if accuracy_model[-1] == max(accuracy_model):
            best_estimator = classifier
            # Calculate the confusion matrix
            c = confusion_matrix(y_test, classifier.predict(X_test))
            # Add the score to the previous confusion matrix of previous
→model

        if isinstance(array, np.ndarray) == False:
            array = c.copy()
        else:
            array = array + c

        # Calculate the confusion matrix
        c = confusion_matrix(y_outliers, classifier.predict(X_outliers))
        # Add the score to the previous confusion matrix of previous
→model

        if isinstance(outliers, np.ndarray) == False:
            outliers = c.copy()
        else:
            outliers = outliers + c

        #Accumulate for classification report
        if isinstance(report_cleaned, list) == False:
            report_cleaned = [y_test, classifier.predict(X_test)]
        else:
            report_cleaned[0] = np.append(report_cleaned[0],y_test)
            report_cleaned[1] = np.append(report_cleaned[1],classifier.
→predict(X_test))

        #Accumulate for classification report
        if isinstance(report_outliers, list) == False:
            report_outliers = [y_outliers, classifier.
→predict(X_outliers)]
        else:
            report_outliers[0] = np.
→append(report_outliers[0],y_outliers)
            report_outliers[1] = np.
→append(report_outliers[1],classifier.predict(X_outliers))

        # Print the accuracy
        print("At split estimator: " + str(n_splits) + ", epochs: " +
→str(epoch))
        print("Accurace mean(std): " + str(np.mean(accuracy_model)) + "(" +
→str(np.std(accuracy_model)) + ")")

        # To calculate the classification reports

```

```

        print("Classification report for all valid cross_validations_
↳against their tests sets")
        print(classification_report(report_cleaned[0], report_cleaned[1],
↳target_names=baseset))

        print("Classification report for all valid cross_validations_
↳against outliers")
        print(classification_report(report_outliers[0], report_outliers[1],
↳target_names=baseset))

        # To calculate the confusion matrix

        print("Confusion Matrix for all valid cross_validations against_
↳their tests sets")
        make_confusion_matrix(array, categories=baseset, figsize=[8,8])

        print("Confusion Matrix for all valid cross_validations against_
↳outliers")
        make_confusion_matrix(outliers, categories=baseset, figsize=[8,8])
    def save_model(model, baseset):
        baseset.sort()
        name = '-'.join(baseset)
        # saving model
        pickle.dump(model.classes_, open(name + '_model_classes.pkl','wb'))
        model.model.save(name + '_lstm')
        save_model(best_estimator, baseset)

model = evaluate_model(baseset)

```

```

Baseset:  ['06', '07', '10']
Loadind Dataset:  ['06', '07', '10']
489 samples loaded
Scaling Dataset:  ['06', '07', '10']
489 samples scaled
Cleaning Dataset:  ['06', '07', '10']
365 samples cleaned
124 samples outliers
Time slicing Cleaned Dataset:  ['06', '07', '10']
364 cleaned samples sliced
1 cleaned samples damaged
Time slicing Outliers Dataset:  ['06', '07', '10']
124 outliers samples sliced
0 outliers samples damaged
Processing started for split estimator: 5, epochs: [64]
Processing 1 -fold
Processing 2 -fold

```

Processing 3 -fold
 Processing 4 -fold
 Processing 5 -fold
 At split estimator: 5, epochs: [64]
 Accurace mean(std): 1.0(0.0)

Classification report for all valid cross_validations against their tests sets

	precision	recall	f1-score	support
06	1.00	1.00	1.00	120
07	1.00	1.00	1.00	119
10	1.00	1.00	1.00	125
accuracy			1.00	364
macro avg	1.00	1.00	1.00	364
weighted avg	1.00	1.00	1.00	364

Classification report for all valid cross_validations against outliers

	precision	recall	f1-score	support
06	0.96	1.00	0.98	220
07	1.00	0.93	0.96	205
10	0.97	1.00	0.98	195
accuracy			0.98	620
macro avg	0.98	0.98	0.98	620
weighted avg	0.98	0.98	0.98	620

Confusion Matrix for all valid cross_validations against their tests sets

Confusion Matrix for all valid cross_validations against outliers

WARNING:absl:Found untraced functions such as lstm_cell_1114_layer_call_fn, lstm_cell_1114_layer_call_and_return_conditional_losses, lstm_cell_1115_layer_call_fn, lstm_cell_1115_layer_call_and_return_conditional_losses, lstm_cell_1114_layer_call_fn while saving (showing 5 of 10). These functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: 06-07-10_lstm/assets

INFO:tensorflow:Assets written to: 06-07-10_lstm/assets

