P3

November 15, 2021

[1]:
```
import os
import pandas as pd

print("Loadind Dataset: gestures")
path = 'gestures-dataset'

dataset = None
dataset_idx = 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])
                    #print(df)
                    if dataset is None:
                        dataset = df.copy()
                    else:
                        dataset = pd.concat([dataset, df])

                    df_idx_sample = pd.DataFrame()
                    df_idx_sample = df_idx_sample.append({'subject':subject,␣
 ↪'gesture': gesture, 'sample': str(samplefile[:-4])}, ignore_index = True)
                    if dataset_idx is None:
                        dataset_idx = df_idx_sample.copy()
                    else:
                        dataset_idx = pd.concat([dataset_idx, df_idx_sample])
                    samples += 1

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

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

scaler = StandardScaler()
dataset_scaled = None
samples = 0
#for i, gesture in enumerate(gesture_subset):
for i, gesture in enumerate(['01', '02', '03', '04', '05', '06', '07', '08',␣
 ↪'09', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20']):
    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
```

```
Loadind Dataset: gestures
3251 samples loaded
Scaling Dataset: gestures
3251 samples scaled
```

[2]:
```
import tsfel

cfg_file = tsfel.get_features_by_domain(domain=None, json_path="features.json")
#cfg_file = tsfel.get_features_by_domain("statistical")
#cfg_file = tsfel.get_features_by_domain("temporal")
#cgf_file = tsfel.get_features_by_domain("spectral")
curr = 127
print("Extracting All Features as describe in features.json, total of 381, 127␣
 ↪per axis: dataset_scaled")
data = dataset_scaled
print("Set: " + str(len(data.index)) + " measurements")
dataset_features = None
samples = 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]
            df_sample.sort_values(by=['sample.timestamp'])

            df_feature = pd.DataFrame(columns = ["gesture","subject", "sample",␣
 ↪"features"])
            #print(df_sample["Y"].size)
            features = {}
            features['X'] =  tsfel.time_series_features_extractor(cfg_file,␣
 ↪df_sample["X"], fs=209, verbose=0)
            features['Y'] =  tsfel.time_series_features_extractor(cfg_file,␣
 ↪df_sample["Y"], fs=209, verbose=0)
            features['Z'] =  tsfel.time_series_features_extractor(cfg_file,␣
 ↪df_sample["Z"], fs=209, verbose=0)
            adj = features['X'].columns.size - 127
            if adj > 0:
                for a in range(adj):
```

```
                    features['X'] = features['X'].drop(labels = ["0_FFT mean␣
↪coefficient_" + str(6+a)], axis = 1)
            if features['X'].columns.size != curr:
                curr = features['X'].columns.size
                print(curr)
                for tmp in features['X'].columns:
                    print(tmp)
            adj = features['Y'].columns.size - 127
            if adj > 0:
                for a in range(adj):
                    features['Y'] = features['Y'].drop(labels = ["0_FFT mean␣
↪coefficient_" + str(6+a)], axis = 1)
            if features['Y'].columns.size != curr:
                curr = features['Y'].columns.size
                print(curr)
                for tmp in features['Y'].columns:
                    print(tmp)
            adj = features['Z'].columns.size - 127
            if adj > 0:
                for a in range(adj):
                    features['Z'] = features['Z'].drop(labels = ["0_FFT mean␣
↪coefficient_" + str(6+a)], axis = 1)
            if features['Z'].columns.size != curr:
                curr = features['Z'].columns.size
                print(curr)
                for tmp in features['Z'].columns:
                    print(tmp)
            df_feature = df_feature.append({ \
                                        'gesture' : gesture, 'subject' :␣
↪subject, 'sample' : sample, \
                                        'features': features
                                       }, \
                                       ignore_index=True)
            if dataset_features is None:
                dataset_features = df_feature.copy()
            else:
                dataset_features = pd.concat([dataset_features, df_feature],␣
↪ignore_index=True)
            samples += 1
#print(dataset_features.head(10))
#print(dataset_features.tail(10))
print("Features extracted from " + str(samples) + " samples")
dataset_scaled_features = dataset_features
```

```
Extracting All Features as describe in features.json, total of 381, 127 per
axis: dataset_scaled
Set: 64070 measurements
```

```
Features extracted from 3251 samples
```

[3]:
```
list = dataset_scaled_features['features'][0]['X'].values.
 ↪tolist()[0]+dataset_scaled_features['features'][0]['Y'].values.
 ↪tolist()[0]+dataset_scaled_features['features'][0]['Z'].values.tolist()[0]
print(type(list))
print(len(list))
```

```
<class 'list'>
381
```

[4]:
```
print("Segregating outliers for testing separately: gestures")
data = dataset_scaled
dataset_outliers = None
dataset_outliers_idx = None
dataset_cleaned = None
dataset_cleaned_idx = None

samples = 0
outliers = 0
#for i, gesture in enumerate(gesture_subset):
for i, gesture in enumerate(['01', '02', '03', '04', '05', '06', '07', '08',␣
 ↪'09', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20']):
    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])
                df_idx_sample = pd.DataFrame()
                df_idx_sample = df_idx_sample.append({'subject':subject,␣
 ↪'gesture': gesture, 'sample': sample}, ignore_index = True)
                if dataset_outliers_idx is None:
```

```
                    dataset_outliers_idx = df_idx_sample.copy()
                else:
                    dataset_outliers_idx = pd.concat([dataset_outliers_idx,␣
 ↪df_idx_sample])
                outliers += 1
            else:
                if dataset_cleaned is None:
                    dataset_cleaned = df_sample.copy()
                else:
                    dataset_cleaned = pd.concat([dataset_cleaned, df_sample])
                df_idx_sample = pd.DataFrame()
                df_idx_sample = df_idx_sample.append({'subject':subject,␣
 ↪'gesture': gesture, 'sample': sample}, ignore_index = True)
                if dataset_cleaned_idx is None:
                    dataset_cleaned_idx = df_idx_sample.copy()
                else:
                    dataset_cleaned_idx = pd.concat([dataset_cleaned_idx,␣
 ↪df_idx_sample])
                samples += 1
print(str(samples) + " samples cleaned")
print(str(outliers) + " samples outliers")
data = dataset_cleaned
```

```
Segregating outliers for testing separately: gestures
2479 samples cleaned
772 samples outliers
```

[5]:
```
print("Segregating a stratified test set in addition to outliers: gestures")
from sklearn.model_selection import train_test_split
data = dataset_cleaned_idx
train, test = train_test_split(data, test_size=0.15, train_size=0.85,␣
 ↪random_state=1000, shuffle=True, stratify=data['gesture'])
print("Training set: " + str(len(train.index)) + " samples")
print("Test set: " + str(len(test.index)) + " samples")
```

```
Segregating a stratified test set in addition to outliers: gestures
Training set: 2107 samples
Test set: 372 samples
```

[6]:
```
print("Filling training samples with measurement data")
data = dataset_cleaned
dataset_cleaned_train = None
samples = 0
for tmp in train.iterrows():
    df = data[data['gesture'] == tmp[1]['gesture']]
    df = df[df['subject'] == tmp[1]['subject']]
    df = df[df['sample'] == tmp[1]['sample']]
    if dataset_cleaned_train is None:
```

```
        dataset_cleaned_train = df.copy()
    else:
        dataset_cleaned_train = pd.concat([dataset_cleaned_train, df])
    samples += 1
print(str(samples) + " training samples filled")

print("Filling test samples with measurement data")
data = dataset_cleaned
dataset_cleaned_test = None
samples = 0
for tmp in test.iterrows():
    df = data[data['gesture'] == tmp[1]['gesture']]
    df = df[df['subject'] == tmp[1]['subject']]
    df = df[df['sample'] == tmp[1]['sample']]
    if dataset_cleaned_test is None:
        dataset_cleaned_test = df.copy()
    else:
        dataset_cleaned_test = pd.concat([dataset_cleaned_test, df])
    samples += 1
print(str(samples) + " test samples filled")
```

```
Filling training samples with measurement data
2107 training samples filled
Filling test samples with measurement data
372 test samples filled
```

[7]:
```
print("Filling training samples with features data")
data = dataset_scaled_features
dataset_cleaned_train_features = None
samples = 0
for tmp in train.iterrows():
    df = data[data['gesture'] == tmp[1]['gesture']]
    df = df[df['subject'] == tmp[1]['subject']]
    df = df[df['sample'] == tmp[1]['sample']]
    if dataset_cleaned_train_features is None:
        dataset_cleaned_train_features = df.copy()
    else:
        dataset_cleaned_train_features = pd.
 ↪concat([dataset_cleaned_train_features, df])
    samples += 1
print(str(samples) + " training samples filled")

print("Filling test samples with feature data")
data = dataset_scaled_features
dataset_cleaned_test_features = None
samples = 0
for tmp in test.iterrows():
```

```
    df = data[data['gesture'] == tmp[1]['gesture']]
    df = df[df['subject'] == tmp[1]['subject']]
    df = df[df['sample'] == tmp[1]['sample']]
    if dataset_cleaned_test_features is None:
        dataset_cleaned_test_features = df.copy()
    else:
        dataset_cleaned_test_features = pd.
 ↪concat([dataset_cleaned_test_features, df])
    samples += 1
print(str(samples) + " test samples filled")

print("Filling outliers samples with feature data")
data = dataset_scaled_features
dataset_outliers_features = None
samples = 0
for tmp in dataset_outliers_idx.iterrows():
    df = data[data['gesture'] == tmp[1]['gesture']]
    df = df[df['subject'] == tmp[1]['subject']]
    df = df[df['sample'] == tmp[1]['sample']]
    if dataset_cleaned_test_features is None:
        dataset_outliers_features = df.copy()
    else:
        dataset_outliers_features = pd.concat([dataset_outliers_features, df])
    samples += 1
print(str(samples) + " outliers samples filled")
```

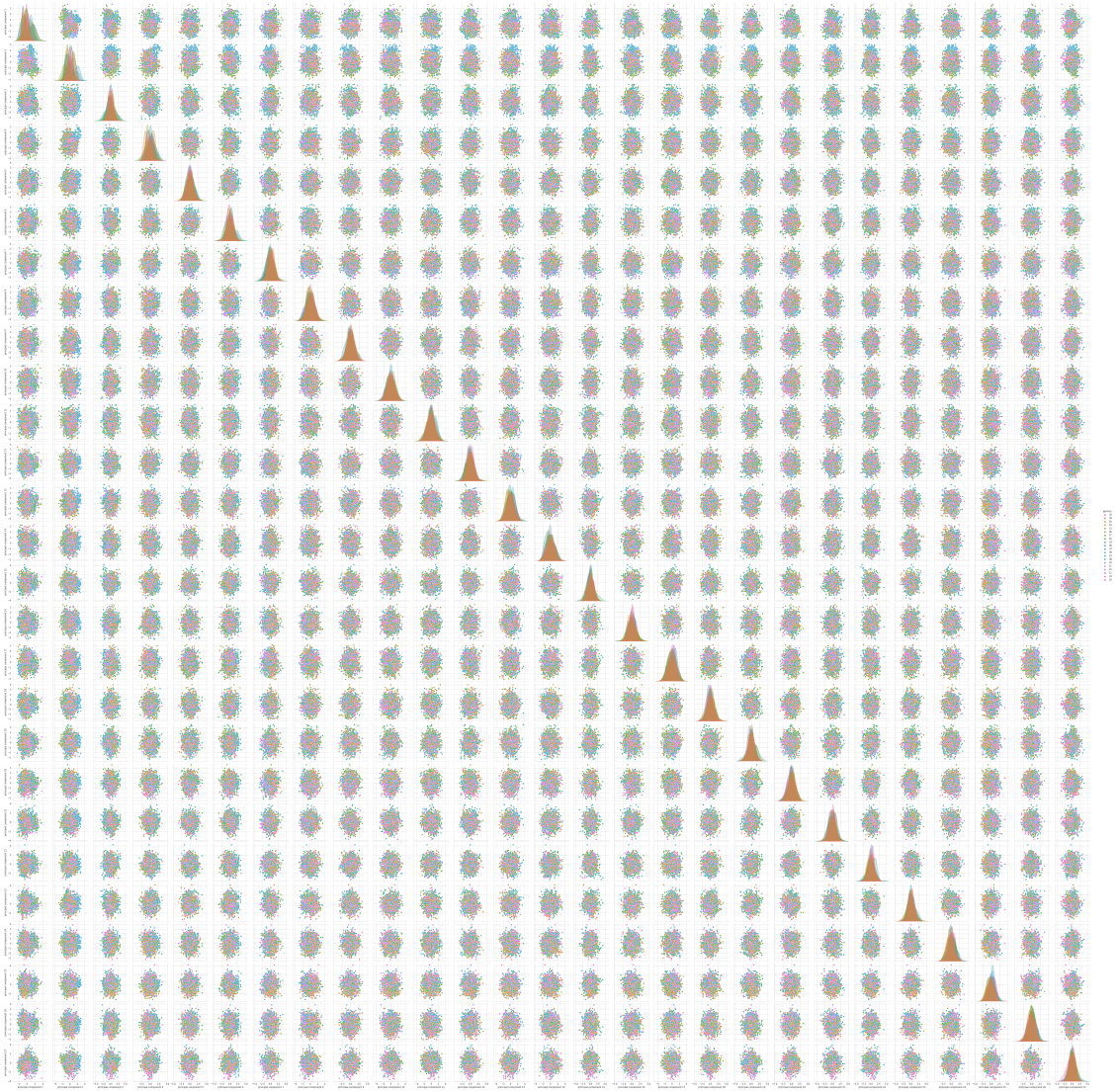
```
Filling training samples with features data
2107 training samples filled
Filling test samples with feature data
372 test samples filled
Filling outliers samples with feature data
772 outliers samples filled
```

[8]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
%matplotlib inline


data = dataset_cleaned_train_features
work = data.copy()
train_targets = pd.DataFrame(work[["gesture","subject","sample"]],␣
 ↪columns=["gesture","subject","sample"])
train_targets.reset_index(inplace=True, drop=True)
target_for_plot = pd.DataFrame(work[["gesture"]], columns=["gesture"])
target_for_plot.reset_index(inplace=True, drop=True)
```

```
work = work.drop(labels = ["gesture","subject", "sample"], axis = 1)
work = work.reset_index(inplace=False, drop=True)
workArray = []
for tmp in work.values:
    tmpList = tmp[0]['X'].values.tolist()[0]+tmp[0]['Y'].values.
 ↪tolist()[0]+tmp[0]['Z'].values.tolist()[0]
    workArray.append(tmpList)

#pca = PCA(n_components=3)
pca = PCA(.95, svd_solver='full', whiten=True)
principalComponents = pca.fit_transform(np.array(workArray))
print(pca.n_features_)
print(pca.n_samples_)
print(pca.n_components_)
print(pca.noise_variance_)

columns = []
for i in range(pca.n_components_):
    columns.append("principal component "+str(i+1))
train_features_pca = pd.DataFrame(data = principalComponents, columns = columns)
train_features_pca = pd.concat([train_features_pca, train_targets], axis = 1)
features_pca_for_plot = pd.DataFrame(data = principalComponents, columns =␣
 ↪columns)
features_pca_for_plot = pd.concat([features_pca_for_plot, target_for_plot],␣
 ↪axis = 1)

fig = plt.figure(figsize = (70,70))
import seaborn as sns

sns.set_style("whitegrid");
#sns.pairplot(iris, hue = "species", size = 3);
#plt.show()
# NOTE: the diagonal elements are PDFs for each feature.import seaborn as sns
sns.pairplot(features_pca_for_plot, hue='gesture')
plt.show()
```

```
381
2107
27
61.699079627611795
```

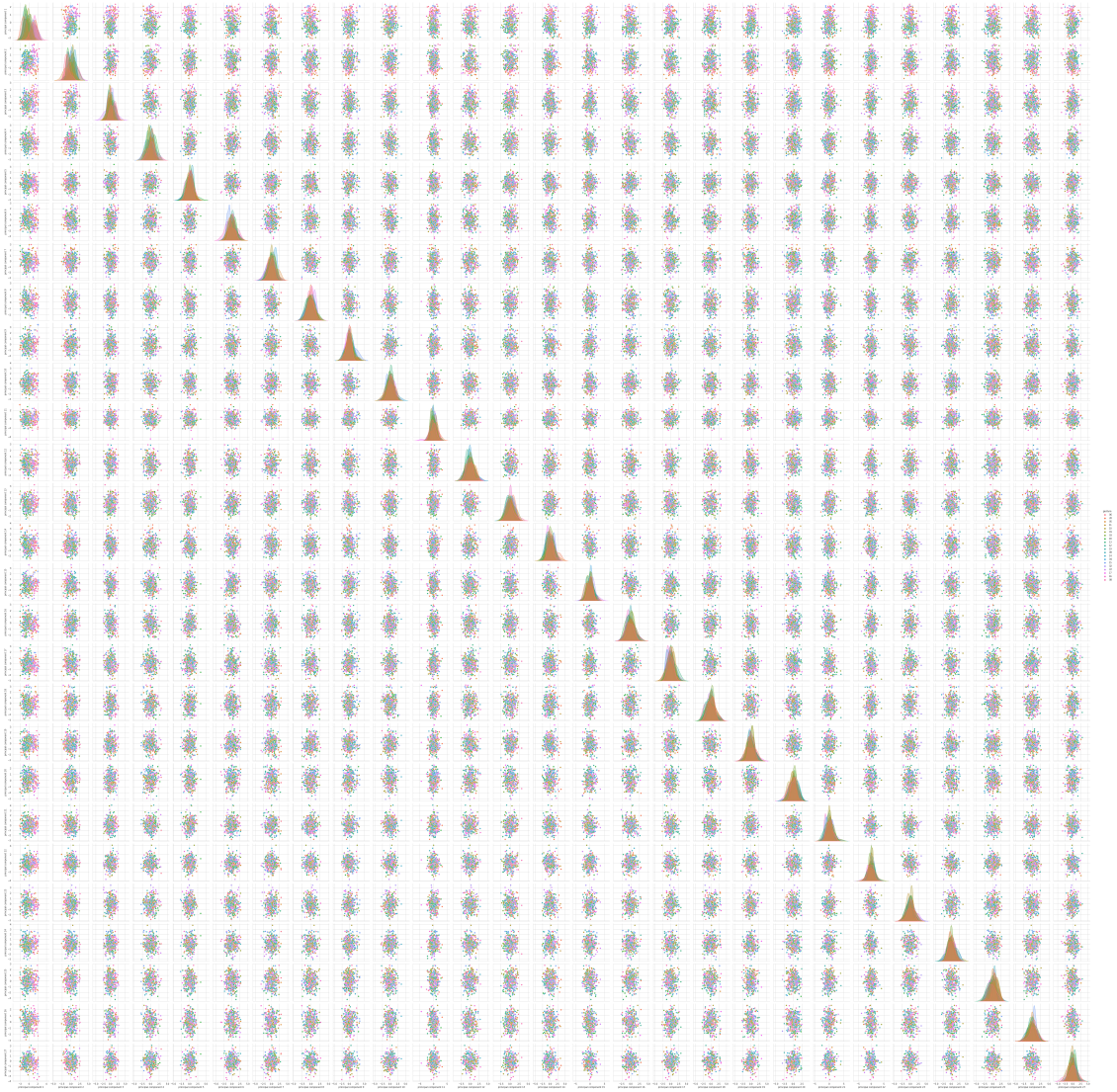
```
<Figure size 5040x5040 with 0 Axes>
```



[9]:
```
data = dataset_cleaned_test_features
work = data.copy()
test_targets = pd.DataFrame(work[["gesture","subject","sample"]],␣
 ↪columns=["gesture","subject","sample"])
test_targets.reset_index(inplace=True, drop=True)
target_for_plot = pd.DataFrame(work[["gesture"]], columns=["gesture"])
target_for_plot.reset_index(inplace=True, drop=True)
work = work.drop(labels = ["gesture","subject", "sample"], axis = 1)
work = work.reset_index(inplace=False, drop=True)
workArray = []
for tmp in work.values:
    tmpList = tmp[0]['X'].values.tolist()[0]+tmp[0]['Y'].values.
 ↪tolist()[0]+tmp[0]['Z'].values.tolist()[0]
```

```
    workArray.append(tmpList)

#pca = PCA(n_components=3)
#pca = PCA(.90)
principalComponents = pca.transform(np.array(workArray))

columns = []
for i in range(pca.n_components_):
    columns.append("principal component "+str(i+1))
test_features_pca = pd.DataFrame(data = principalComponents, columns = columns)
test_features_pca = pd.concat([test_features_pca, test_targets], axis = 1)
features_pca_for_plot = pd.DataFrame(data = principalComponents, columns =␣
 ↪columns)
features_pca_for_plot = pd.concat([features_pca_for_plot, target_for_plot],␣
 ↪axis = 1)

fig = plt.figure(figsize = (70,70))
import seaborn as sns

sns.set_style("whitegrid");
#sns.pairplot(iris, hue = "species", size = 3);
#plt.show()
# NOTE: the diagonal elements are PDFs for each feature.import seaborn as sns
sns.pairplot(features_pca_for_plot, hue='gesture')
plt.show()
```

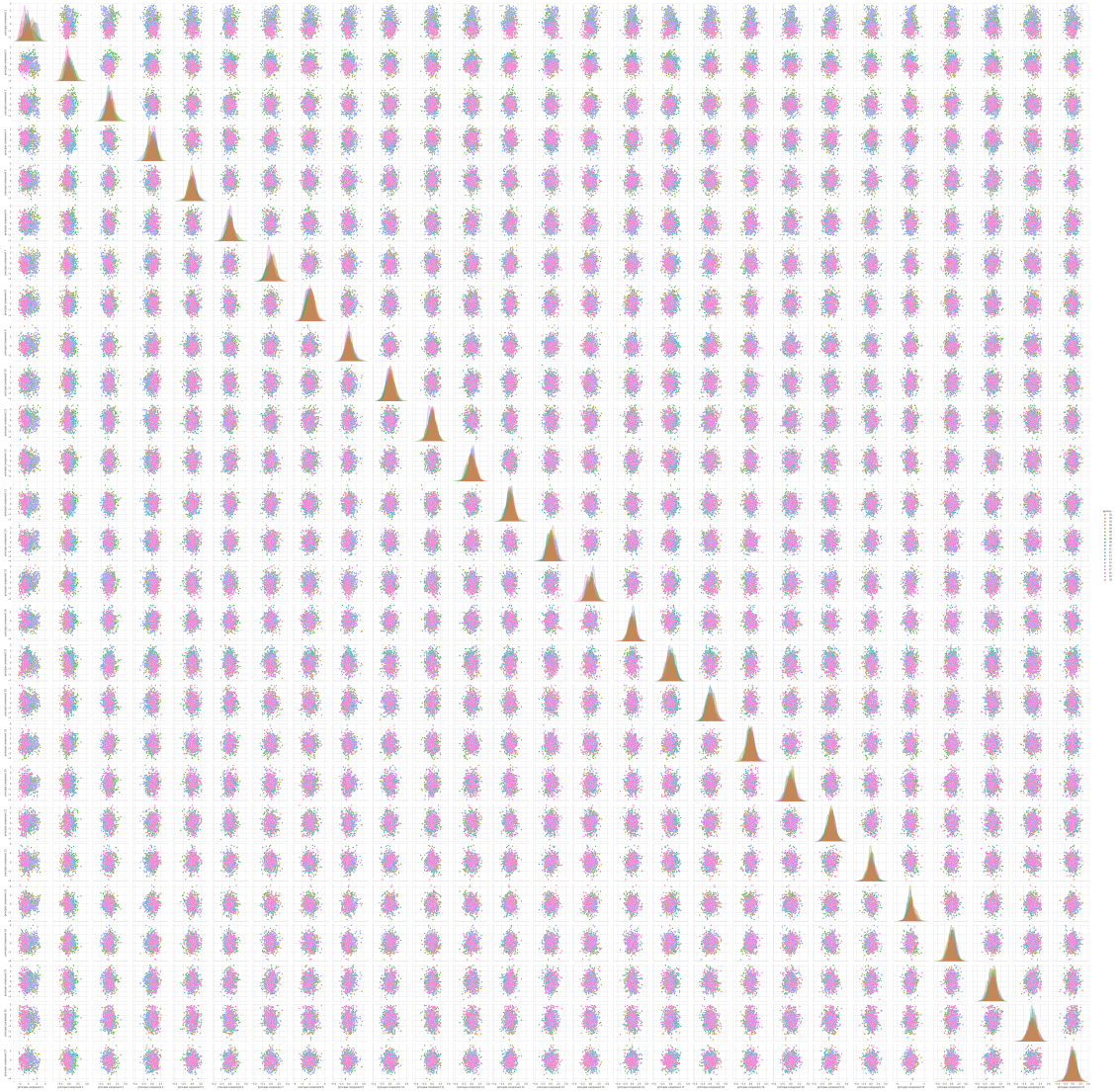
```
<Figure size 5040x5040 with 0 Axes>
```



[10]:
```
data = dataset_outliers_features
work = data.copy()
outliers_targets = pd.DataFrame(work[["gesture","subject","sample"]],␣
 ↪columns=["gesture","subject","sample"])
outliers_targets.reset_index(inplace=True, drop=True)
target_for_plot = pd.DataFrame(work[["gesture"]], columns=["gesture"])
target_for_plot.reset_index(inplace=True, drop=True)
work = work.drop(labels = ["gesture","subject", "sample"], axis = 1)
work = work.reset_index(inplace=False, drop=True)
workArray = []
for tmp in work.values:
    tmpList = tmp[0]['X'].values.tolist()[0]+tmp[0]['Y'].values.
 ↪tolist()[0]+tmp[0]['Z'].values.tolist()[0]
```

```
    workArray.append(tmpList)

#pca = PCA(n_components=3)
#pca = PCA(.90)
principalComponents = pca.transform(np.array(workArray))

columns = []
for i in range(pca.n_components_):
    columns.append("principal component "+str(i+1))
outliers_features_pca = pd.DataFrame(data = principalComponents, columns =␣
 ↪columns)
outliers_features_pca = pd.concat([outliers_features_pca, outliers_targets],␣
 ↪axis = 1)
features_pca_for_plot = pd.DataFrame(data = principalComponents, columns =␣
 ↪columns)
features_pca_for_plot = pd.concat([features_pca_for_plot, target_for_plot],␣
 ↪axis = 1)

fig = plt.figure(figsize = (70,70))
import seaborn as sns

sns.set_style("whitegrid");
#sns.pairplot(iris, hue = "species", size = 3);
#plt.show()
# NOTE: the diagonal elements are PDFs for each feature.import seaborn as sns
sns.pairplot(features_pca_for_plot, hue='gesture')
plt.show()
```

```
<Figure size 5040x5040 with 0 Axes>
```



[11]:
```
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.models import Model
from keras.layers import Input
from keras.layers import LeakyReLU
from keras.layers import BatchNormalization
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_val_score
```

```
from sklearn.model_selection import GridSearchCV
from keras.utils import np_utils
from keras.utils.vis_utils import plot_model
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)
```

[12]:
```
# create the dataset
def get_dataset_features(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]
                df_sample = df_sample.drop(labels = ["gesture","subject",␣
 ↪"sample"], axis = 1)
                df_sample = df_sample.reset_index(inplace=False, drop=True)
                workArray = []
                for tmp in df_sample.values:
                    tmpList = tmp[0]['X'].values.tolist()[0]+tmp[0]['Y'].values.
 ↪tolist()[0]+tmp[0]['Z'].values.tolist()[0]
                    #print(tmpList)
                    workArray.append(tmpList)
                #print(df_sample.values[0])
                X_train.append(np.asarray(workArray[0]))
                Y_train.append(gesture)
                groups.append(subject)
    X_train = np.asarray(X_train)
    Y_train = LabelEncoder().fit_transform(Y_train)
    return X_train, Y_train, groups
```

[13]:
```
X, y, g = get_dataset_features(dataset_cleaned_train_features)
#print(X[1])
```

[14]:
```
# Function to create model, required for KerasClassifier
def create_model(n_inputs=24, dropout_rate=0.8, units=128, optimizer=adam_v2.
 ↪Adam(learning_rate=0.001)):

    model = Sequential()
```

```
    model.add(Input(shape=(n_inputs,)))
    # encoder level 1
    model.add(Dense(2*n_inputs))
    model.add(BatchNormalization())
    model.add(LeakyReLU())
    # encoder level 2
    model.add(Dense(n_inputs))
    model.add(BatchNormalization())
    model.add(LeakyReLU())
    # bottleneck
    n_bottleneck = round(float(n_inputs) / dropout_rate)
    bottleneck = Dense(n_bottleneck)
    model.add(bottleneck)
    # define decoder, level 1
    model.add(Dense(n_inputs))
    model.add(BatchNormalization())
    model.add(LeakyReLU())
    # decoder level 2
    model.add(Dense(2*n_inputs))
    model.add(BatchNormalization())
    model.add(LeakyReLU())
    # output layer
    model.add(Dense(20, activation='softmax'))
    model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer,␣
↪metrics=['accuracy'])
    #model.build((n_inputs,))
    #print(model.summary())
    return model
```

[15]:
```
print("Training an autoencoder model with the 381 direct statistical features,␣
 ↪gridsearch over bottleneck")
data=dataset_cleaned_train_features
X, y, g = get_dataset_features(data)
#n_inputs = X.shape[0]
#print(n_inputs)
n_inputs = X.shape[1]
n_inputs = [n_inputs]
#print(n_inputs)
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_features(data)
#cv = cv.split(X, y, g)
batch_size = [19]
#epochs = [64, 128]
epochs = [128]
#units = [32,64,128]
```

```
units = [128]
#dropout_rate = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
dropout_rate = [1.0, 2.0, 3.0, 6.0, 9.0]
param_grid = dict(n_inputs=n_inputs,epochs=epochs, units=units,␣
 ↪batch_size=batch_size, dropout_rate=dropout_rate)
#print("Hyperparameter tunning started for Dataset for gestures: ",␣
 ↪gesture_subset)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1, cv=cv,␣
 ↪verbose=1, error_score='raise')
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("Training and tunning completed for Dataset: gestures")
```

```
Training an autoencoder model with the 381 direct statistical features,
gridsearch over bottleneck
Fitting 5 folds for each of 5 candidates, totalling 25 fits

2021-11-15 11:34:39.767270: 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.
2021-11-15 11:34:39.927065: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR
Optimization Passes are enabled (registered 2)

Best: 0.634647 using {'batch_size': 19, 'dropout_rate': 3.0, 'epochs': 128,
'n_inputs': 381, 'units': 128}
accuracy: 0.597801 (0.082041) train time: 49.041778 (4.735642) score time:
0.281400 (0.134938) with: {'batch_size': 19, 'dropout_rate': 1.0, 'epochs': 128,
'n_inputs': 381, 'units': 128}
accuracy: 0.582342 (0.072800) train time: 46.159583 (4.153968) score time:
0.212230 (0.010722) with: {'batch_size': 19, 'dropout_rate': 2.0, 'epochs': 128,
'n_inputs': 381, 'units': 128}
accuracy: 0.634647 (0.054616) train time: 44.944802 (4.009744) score time:
```

```
0.209278 (0.011437) with: {'batch_size': 19, 'dropout_rate': 3.0, 'epochs': 128,
'n_inputs': 381, 'units': 128}
accuracy: 0.600170 (0.061464) train time: 41.966161 (4.056529) score time:
0.203042 (0.008784) with: {'batch_size': 19, 'dropout_rate': 6.0, 'epochs': 128,
'n_inputs': 381, 'units': 128}
accuracy: 0.595186 (0.074417) train time: 39.929319 (3.596336) score time:
0.265362 (0.127604) with: {'batch_size': 19, 'dropout_rate': 9.0, 'epochs': 128,
'n_inputs': 381, 'units': 128}
Training and tunning completed for Dataset: gestures
```

[16]:
```
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 + '_auto')
print("Saving model to disk started for Dataset: gestures")
save_model(model, ["Autoencoder","Features"])
print("Saving model to disk completed for Dataset: gestures")

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 + '_auto')
    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")
model = load_model(["Autoencoder","Features"])
print(model.model.summary())
print("Loading model to disk completed for Dataset: gestures")
```

```
Saving model to disk started for Dataset: gestures

2021-11-15 11:54:09.566412: 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.

INFO:tensorflow:Assets written to: Autoencoder-Features_auto/assets
Saving model to disk completed for Dataset: gestures
Loading model to disk started for Dataset: gestures
Model: "sequential_25"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_150 (Dense)            (None, 762)               291084
_________________________________________________________________
batch_normalization_100 (Bat (None, 762)               3048
_________________________________________________________________
leaky_re_lu_100 (LeakyReLU)  (None, 762)               0
_________________________________________________________________
dense_151 (Dense)            (None, 381)               290703
_________________________________________________________________
batch_normalization_101 (Bat (None, 381)               1524
_________________________________________________________________
leaky_re_lu_101 (LeakyReLU)  (None, 381)               0
_________________________________________________________________
dense_152 (Dense)            (None, 127)               48514
_________________________________________________________________
dense_153 (Dense)            (None, 381)               48768
_________________________________________________________________
batch_normalization_102 (Bat (None, 381)               1524
_________________________________________________________________
leaky_re_lu_102 (LeakyReLU)  (None, 381)               0
_________________________________________________________________
dense_154 (Dense)            (None, 762)               291084
_________________________________________________________________
batch_normalization_103 (Bat (None, 762)               3048
_________________________________________________________________
leaky_re_lu_103 (LeakyReLU)  (None, 762)               0
_________________________________________________________________
dense_155 (Dense)            (None, 20)                15260
=================================================================
Total params: 994,557
Trainable params: 989,985
Non-trainable params: 4,572
_________________________________________________________________
None
Loading model to disk completed for Dataset: gestures
```

[17]:
```
print("Testing model against test set for Dataset: gestures")
data = dataset_cleaned_test_features
X, y, g = get_dataset_features(data)
y_pred = model.predict(X.tolist())
from sklearn.metrics import classification_report
print(classification_report(y, y_pred))

#from sklearn.metrics import confusion_matrix
#cf_matrix = confusion_matrix(y, y_pred)
```

```
#make_confusion_matrix(cf_matrix, categories=gesture_subset, figsize=[8,8])
```

```
Testing model against test set for Dataset: gestures
              precision    recall  f1-score   support

           0       0.71      0.83      0.77        18
           1       0.80      0.63      0.71        19
           2       0.90      0.90      0.90        20
           3       1.00      0.95      0.97        19
           4       0.83      0.88      0.86        17
           5       0.84      0.89      0.86        18
           6       1.00      0.89      0.94        18
           7       0.90      0.95      0.92        19
           8       0.86      0.95      0.90        19
           9       0.86      0.95      0.90        19
          10       0.84      0.89      0.86        18
          11       0.93      0.78      0.85        18
          12       0.73      0.80      0.76        20
          13       0.82      0.70      0.76        20
          14       0.71      0.94      0.81        18
          15       0.91      0.56      0.69        18
          16       0.90      1.00      0.95        18
          17       0.95      0.95      0.95        20
          18       0.88      0.88      0.88        17
          19       0.94      0.89      0.92        19

    accuracy                           0.86       372
   macro avg       0.87      0.86      0.86       372
weighted avg       0.87      0.86      0.86       372
```

[18]:
```
print("Testing model against outliers set for Dataset: gestures")
data = dataset_outliers_features
X, y, g = get_dataset_features(data)
y_pred = model.predict(X.tolist())
from sklearn.metrics import classification_report
print(classification_report(y, y_pred))

#from sklearn.metrics import confusion_matrix
#cf_matrix = confusion_matrix(y, y_pred)
#make_confusion_matrix(cf_matrix, categories=gesture_subset, figsize=[8,8])
```

```
Testing model against outliers set for Dataset: gestures
              precision    recall  f1-score   support

           0       0.66      0.64      0.65        39
           1       0.68      0.42      0.52        36
           2       0.91      0.59      0.71        34
```

```
           3       0.88      0.88      0.88        42
           4       0.81      0.68      0.74        44
           5       0.81      0.77      0.79        44
           6       0.88      0.73      0.80        41
           7       0.46      0.97      0.63        38
           8       0.71      0.94      0.81        32
           9       0.79      0.95      0.86        39
          10       0.70      0.71      0.71        42
          11       0.71      0.62      0.67        40
          12       0.64      0.62      0.63        26
          13       0.79      0.72      0.75        32
          14       0.78      0.91      0.84        44
          15       0.76      0.66      0.71        44
          16       0.97      0.68      0.80        41
          17       0.81      0.72      0.76        29
          18       0.76      0.81      0.78        47
          19       0.78      0.82      0.79        38

    accuracy                           0.75       772
   macro avg       0.76      0.74      0.74       772
weighted avg       0.77      0.75      0.75       772
```

[19]:
```
# create the dataset
def get_dataset_features_pca(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]
                df_sample = df_sample.drop(labels = ["gesture","subject",␣
 ↪"sample"], axis = 1)
                #print(df_sample.values[0])
                X_train.append(np.asarray(df_sample.values[0]))
                Y_train.append(gesture)
                groups.append(subject)
    X_train = np.asarray(X_train)
    #print(X_train)
    Y_train = LabelEncoder().fit_transform(Y_train)
    return X_train, Y_train, groups
```

[20]:
```
X, y, g = get_dataset_features_pca(train_features_pca)
#print(X)
```

[21]:
```
print("Training an autoencoder model with the PCA features, 1x girth because␣
 ↪PCA is already compressed")
data=train_features_pca
X, y, g = get_dataset_features_pca(data)
n_inputs = X.shape[1]
n_inputs = [n_inputs]
#print(n_inputs)
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_features_pca(data)
#cv = cv.split(X, y, g)
batch_size = [19]
#epochs = [64, 128]
epochs = [128]
#units = [32,64,128]
units = [128]
#dropout_rate = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
dropout_rate = [1.0]
param_grid = dict(n_inputs=n_inputs,epochs=epochs, units=units,␣
 ↪batch_size=batch_size, dropout_rate=dropout_rate)
#print("Hyperparameter tunning 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("Training and tunning completed for Dataset: gestures")
```

```
Training an autoencoder model with the PCA features, 1x girth because PCA is
already compressed
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Best: 0.408233 using {'batch_size': 19, 'dropout_rate': 1.0, 'epochs': 128,
'n_inputs': 27, 'units': 128}
accuracy: 0.408233 (0.056236) train time: 16.722026 (1.331812) score time:
```

```
0.260476 (0.159935) with: {'batch_size': 19, 'dropout_rate': 1.0, 'epochs': 128,
'n_inputs': 27, 'units': 128}
Training and tunning completed for Dataset: gestures
```

[22]:
```
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 + '_auto')
print("Saving model to disk started for Dataset: gestures")
save_model(model, ["Autoencoder","Features", "PCA"])
print("Saving model to disk completed for Dataset: gestures")

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 + '_auto')
    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")
model = load_model(["Autoencoder","Features", "PCA"])
print(model.model.summary())
print("Loading model to disk completed for Dataset: gestures")
```

```
Saving model to disk started for Dataset: gestures
INFO:tensorflow:Assets written to: Autoencoder-Features-PCA_auto/assets
Saving model to disk completed for Dataset: gestures
Loading model to disk started for Dataset: gestures
Model: "sequential_31"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_186 (Dense)            (None, 54)                1512
_________________________________________________________________
batch_normalization_124 (Bat (None, 54)                216
_________________________________________________________________
leaky_re_lu_124 (LeakyReLU)  (None, 54)                0
_________________________________________________________________
dense_187 (Dense)            (None, 27)                1485
```

```
_________________________________________________________________
batch_normalization_125 (Bat (None, 27)                108
_________________________________________________________________
leaky_re_lu_125 (LeakyReLU)  (None, 27)                0
_________________________________________________________________
dense_188 (Dense)            (None, 27)                756
_________________________________________________________________
dense_189 (Dense)            (None, 27)                756
_________________________________________________________________
batch_normalization_126 (Bat (None, 27)                108
_________________________________________________________________
leaky_re_lu_126 (LeakyReLU)  (None, 27)                0
_________________________________________________________________
dense_190 (Dense)            (None, 54)                1512
_________________________________________________________________
batch_normalization_127 (Bat (None, 54)                216
_________________________________________________________________
leaky_re_lu_127 (LeakyReLU)  (None, 54)                0
_________________________________________________________________
dense_191 (Dense)            (None, 20)                1100
=================================================================
Total params: 7,769
Trainable params: 7,445
Non-trainable params: 324
_________________________________________________________________
None
Loading model to disk completed for Dataset: gestures
```

[23]:
```
print("Testing model against test set for Dataset: gestures")
data = test_features_pca
X, y, g = get_dataset_features_pca(data)
y_pred = model.predict(X.tolist())
from sklearn.metrics import classification_report
print(classification_report(y, y_pred))

#from sklearn.metrics import confusion_matrix
#cf_matrix = confusion_matrix(y, y_pred)
#make_confusion_matrix(cf_matrix, categories=gesture_subset, figsize=[8,8])
```

```
Testing model against test set for Dataset: gestures
              precision    recall  f1-score   support

           0       0.29      0.22      0.25        18
           1       0.47      0.37      0.41        19
           2       0.70      0.70      0.70        20
           3       0.57      0.68      0.62        19
           4       0.84      0.94      0.89        17
           5       0.78      0.78      0.78        18
```

```
           6       0.88      0.83      0.86        18
           7       0.84      0.84      0.84        19
           8       0.61      0.74      0.67        19
           9       0.84      0.84      0.84        19
          10       0.79      0.61      0.69        18
          11       0.56      0.50      0.53        18
          12       0.75      0.75      0.75        20
          13       0.65      0.55      0.59        20
          14       0.75      0.83      0.79        18
          15       0.68      0.72      0.70        18
          16       0.77      0.56      0.65        18
          17       0.68      0.85      0.76        20
          18       0.65      0.65      0.65        17
          19       0.71      0.89      0.79        19

    accuracy                           0.69       372
   macro avg       0.69      0.69      0.69       372
weighted avg       0.69      0.69      0.69       372
```

[24]:
```
print("Testing model against outliers set for Dataset: gestures")
data = outliers_features_pca
X, y, g = get_dataset_features_pca(data)
y_pred = model.predict(X.tolist())
from sklearn.metrics import classification_report
print(classification_report(y, y_pred))

#from sklearn.metrics import confusion_matrix
#cf_matrix = confusion_matrix(y, y_pred)
#make_confusion_matrix(cf_matrix, categories=gesture_subset, figsize=[8,8])
```

```
Testing model against outliers set for Dataset: gestures
              precision    recall  f1-score   support

           0       0.37      0.38      0.37        39
           1       0.32      0.33      0.33        36
           2       0.72      0.68      0.70        34
           3       0.55      0.43      0.48        42
           4       0.71      0.55      0.62        44
           5       0.54      0.73      0.62        44
           6       0.73      0.78      0.75        41
           7       0.74      0.68      0.71        38
           8       0.62      0.56      0.59        32
           9       0.68      0.72      0.70        39
          10       0.65      0.71      0.68        42
          11       0.56      0.45      0.50        40
          12       0.46      0.50      0.48        26
          13       0.57      0.53      0.55        32
```

```
           14       0.65      0.73      0.69        44
           15       0.65      0.55      0.59        44
           16       0.54      0.51      0.53        41
           17       0.46      0.62      0.53        29
           18       0.62      0.62      0.62        47
           19       0.62      0.66      0.64        38

     accuracy                           0.59       772
    macro avg       0.59      0.59      0.58       772
 weighted avg       0.59      0.59      0.59       772
```

[25]:
```
print("Time slicing Cleaned Dataset for gestures: dataset_cleaned_train")
data=dataset_cleaned_train
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_train_timecut = dataset_timecut
print(str(samples) + " cleaned samples sliced")
print(str(damaged) + " cleaned samples damaged")
```

```
Time slicing Cleaned Dataset for gestures: dataset_cleaned_train
2106 cleaned samples sliced
1 cleaned samples damaged
```

[26]:
```
print("Time slicing Cleaned Dataset for gestures: dataset_cleaned_test")
data=dataset_cleaned_test
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_test_timecut = dataset_timecut
print(str(samples) + " cleaned samples sliced")
print(str(damaged) + " cleaned samples damaged")
```

```
Time slicing Cleaned Dataset for gestures: dataset_cleaned_test
372 cleaned samples sliced
0 cleaned samples damaged
```

[27]:
```
print("Time slicing Cleaned Dataset for gestures: dataset_outliers")
data=dataset_outliers
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_timecut = dataset_timecut
print(str(samples) + " outliers samples sliced")
```

```
print(str(damaged) + " outliers samples damaged")
```

```
Time slicing Cleaned Dataset for gestures: dataset_outliers
769 outliers samples sliced
3 outliers samples damaged
```

[31]:
```
# 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(np.
 ↪asarray([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
```

[32]:
```
# 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(20, activation='softmax'))
```

```
    model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer,␣
 ↪metrics=['accuracy'])
    #print(model.summary())
    return model
```

[33]:
```
data = dataset_cleaned_train_timecut
print("Training a LSTM model from measurement data for comparison")

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(data)
#cv = cv.split(X, y, g)
batch_size = [19]
#epochs = [64, 128]
epochs = [128]
#units = [32,64,128]
units = [128]
#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 tunning 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("Training and tunning completed for Dataset: gestures")
```

```
Training a LSTM model from measurement data for comparison
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Best: 0.934965 using {'batch_size': 19, 'dropout_rate': 0.5, 'epochs': 128,
'units': 128}
accuracy: 0.934965 (0.039706) train time: 131.592410 (12.340437) score time:
```

```
0.799166 (0.150575) with: {'batch_size': 19, 'dropout_rate': 0.5, 'epochs': 128,
'units': 128}
Training and tunning completed for Dataset: gestures
```

[34]:
```
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")
save_model(model, ["LSTM","Measurements"])
print("Saving model to disk completed for Dataset: gestures")

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")
model = load_model(["LSTM","Measurements"])
print(model.model.summary())
print("Loading model to disk completed for Dataset: gestures")
```

```
Saving model to disk started for Dataset: gestures

WARNING:absl:Found untraced functions such as lstm_cell_34_layer_call_fn,
lstm_cell_34_layer_call_and_return_conditional_losses,
lstm_cell_35_layer_call_fn,
lstm_cell_35_layer_call_and_return_conditional_losses,
lstm_cell_34_layer_call_fn while saving (showing 5 of 10). These functions will
not be directly callable after loading.

INFO:tensorflow:Assets written to: LSTM-Measurements_lstm/assets

INFO:tensorflow:Assets written to: LSTM-Measurements_lstm/assets

Saving model to disk completed for Dataset: gestures
Loading model to disk started for Dataset: gestures
Model: "sequential_43"

_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_11 (Bidirectio (None, 256)               135168
_________________________________________________________________
dropout_11 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_214 (Dense)            (None, 128)               32896
_________________________________________________________________
dense_215 (Dense)            (None, 20)                2580
=================================================================
Total params: 170,644
Trainable params: 170,644
Non-trainable params: 0
_________________________________________________________________
None
Loading model to disk completed for Dataset: gestures
```

[35]:
```
print("Testing model against test set for Dataset: gestures")
data = dataset_cleaned_test_timecut
X, y, g = get_dataset(data)
y_pred = model.predict(X.tolist())
from sklearn.metrics import classification_report
print(classification_report(y, y_pred))

#from sklearn.metrics import confusion_matrix
#cf_matrix = confusion_matrix(y, y_pred)
#make_confusion_matrix(cf_matrix, categories=gesture_subset, figsize=[8,8])
```

```
Testing model against test set for Dataset: gestures
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        18
           1       1.00      0.95      0.97        19
           2       1.00      1.00      1.00        20
           3       1.00      1.00      1.00        19
           4       1.00      1.00      1.00        17
           5       1.00      0.94      0.97        18
           6       1.00      1.00      1.00        18
           7       1.00      1.00      1.00        19
           8       0.95      1.00      0.97        19
           9       1.00      1.00      1.00        19
          10       1.00      1.00      1.00        18
          11       1.00      1.00      1.00        18
          12       1.00      0.95      0.97        20
          13       0.95      1.00      0.98        20
          14       1.00      1.00      1.00        18
          15       1.00      1.00      1.00        18
          16       1.00      1.00      1.00        18
```

```
           17       1.00      1.00      1.00        20
           18       0.94      0.88      0.91        17
           19       0.90      1.00      0.95        19

     accuracy                           0.99       372
    macro avg       0.99      0.99      0.99       372
 weighted avg       0.99      0.99      0.99       372
```

[36]:
```
print("Testing model against outliers set for Dataset: gestures")
data = dataset_outliers_timecut
X, y, g = get_dataset(data)
y_pred = model.predict(X.tolist())
from sklearn.metrics import classification_report
print(classification_report(y, y_pred))

#from sklearn.metrics import confusion_matrix
#cf_matrix = confusion_matrix(y, y_pred)
#make_confusion_matrix(cf_matrix, categories=gesture_subset, figsize=[8,8])
```

```
Testing model against outliers set for Dataset: gestures
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        39
           1       1.00      1.00      1.00        34
           2       0.97      0.91      0.94        34
           3       1.00      0.98      0.99        42
           4       1.00      0.95      0.98        43
           5       1.00      1.00      1.00        44
           6       1.00      0.98      0.99        41
           7       1.00      0.97      0.99        38
           8       1.00      0.97      0.98        32
           9       1.00      1.00      1.00        39
          10       0.95      1.00      0.98        42
          11       1.00      0.97      0.99        40
          12       1.00      1.00      1.00        26
          13       0.97      1.00      0.98        32
          14       0.98      1.00      0.99        44
          15       0.98      1.00      0.99        44
          16       1.00      1.00      1.00        41
          17       0.93      0.97      0.95        29
          18       0.98      0.98      0.98        47
          19       0.93      1.00      0.96        38

    accuracy                           0.98       769
   macro avg       0.98      0.98      0.98       769
weighted avg       0.98      0.98      0.98       769
```

[ ]: