Group Members: Edgar Pino and Gassan Soukaev
CS498 Applied Machine Learning

CS498 AMO

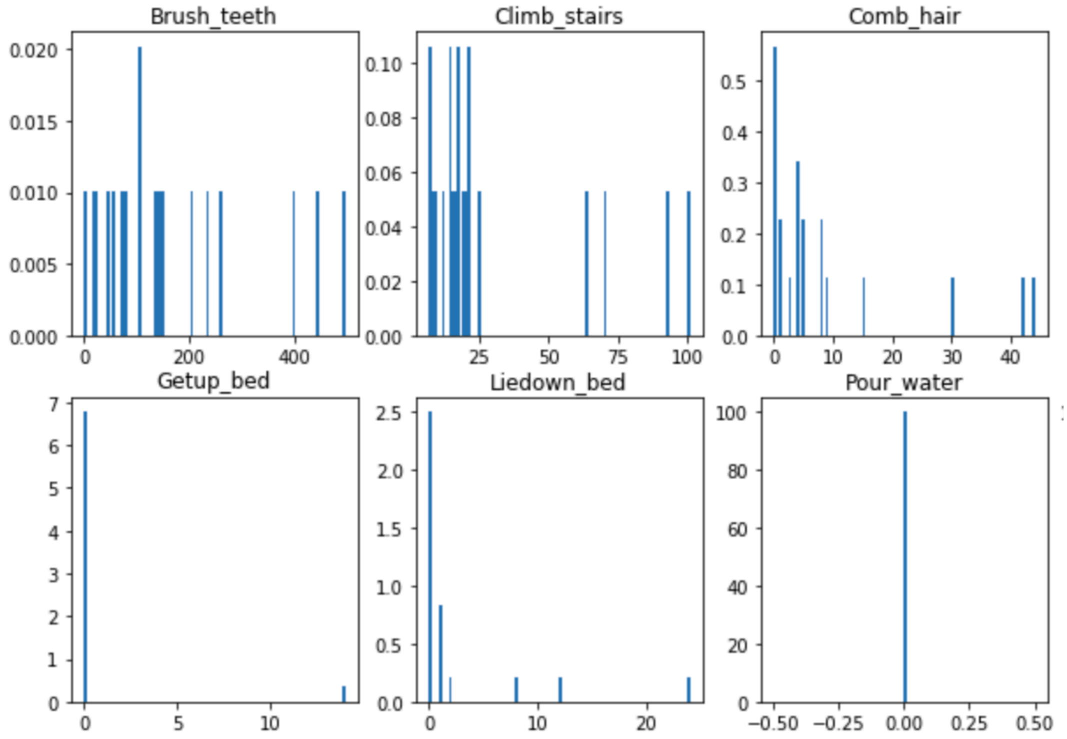
Experiment table

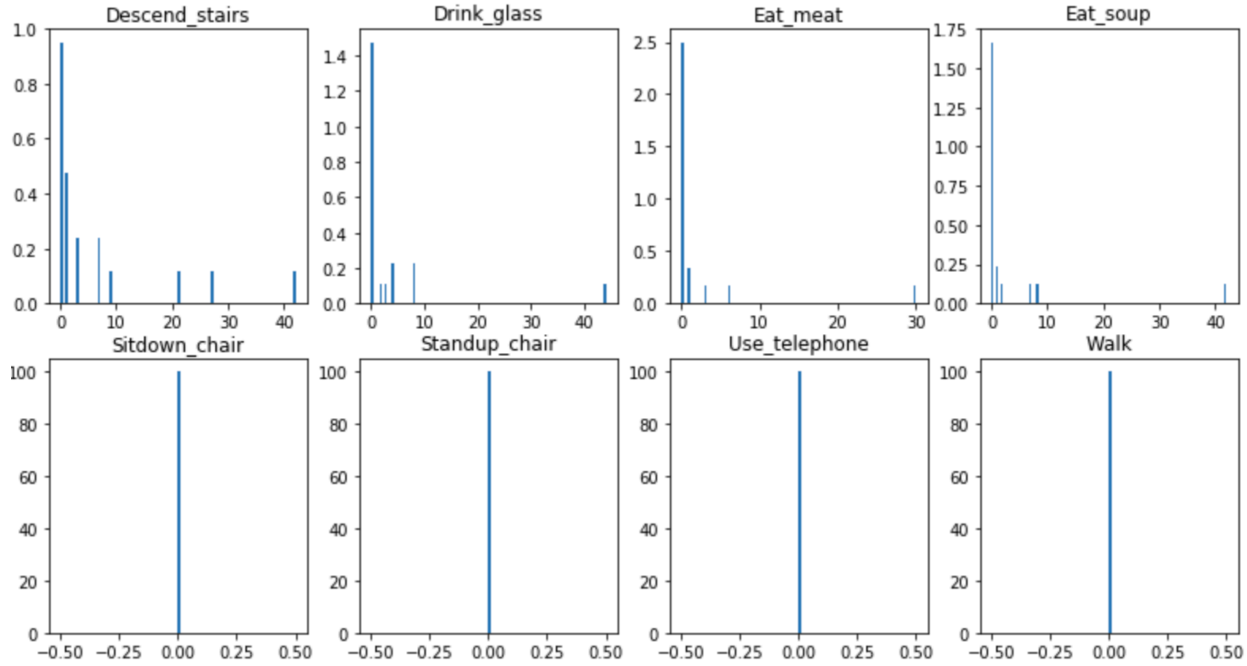
K-Value	Lengths	Accuracy
16	1	75.8993
20	1	81.295
25	1	75.8993
30	1	71.223
40	1	78.777
16	4	75.8993
20	4	75.1799
25	4	76.9784
30	4	73.741
40	4	75.5396
16	8	71.223
20	8	74.8201
25	8	74.4604
30	8	73.0216
40	8	74.4604

Used standard K-means

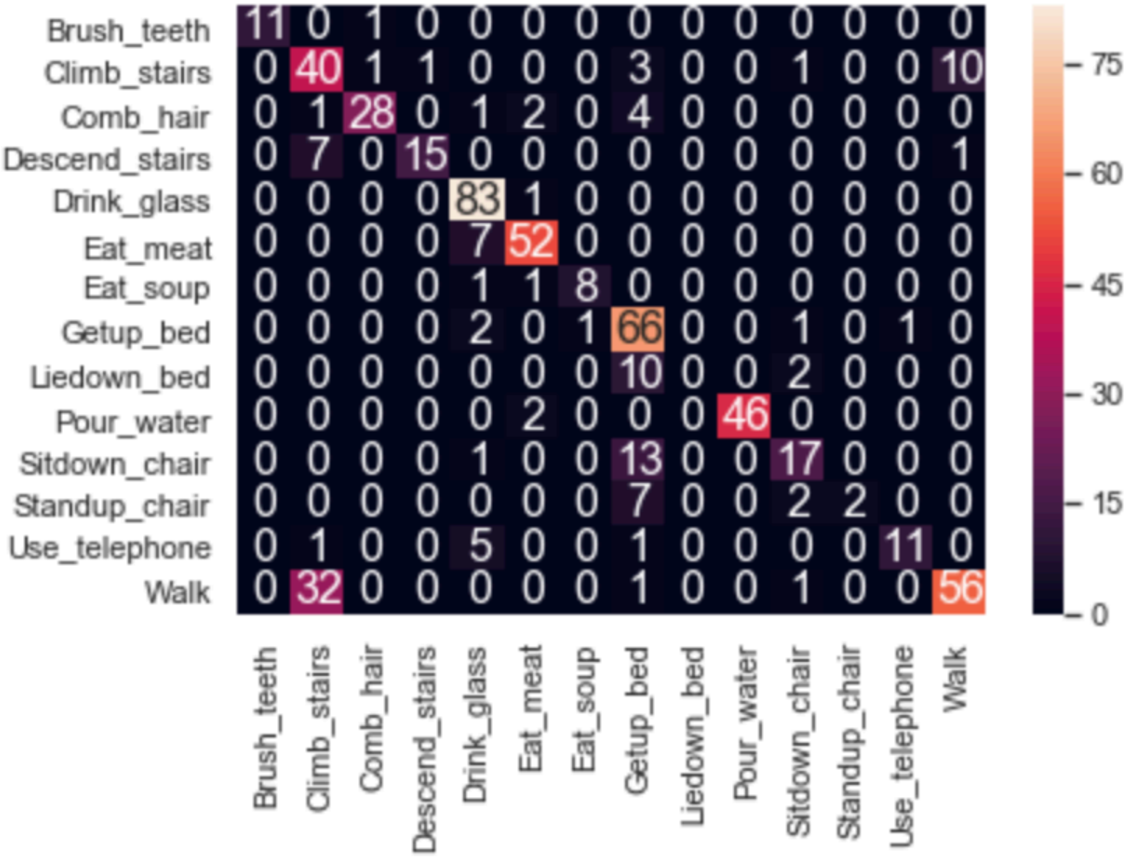
Histograms

K-Value = 20





Confusion matrix



	Brush_teeth	Climb_stairs	Comb_hair	Descend_stairs	Drink_glass	Eat_meat	Eat_soup	Getup_bed	Liedown_bed	Pour_water	Sitdown_chair	Standup_chair	Use_telephone	Walk
Brush_teeth	11	0	1	0	0	0	0	0	0	0	0	0	0	0
Climb_stairs	0	40	1	1	0	0	0	3	0	0	1	0	0	10
Comb_hair	0	1	28	0	1	2	0	4	0	0	0	0	0	0
Descend_stairs	0	7	0	15	0	0	0	0	0	0	0	0	0	1
Drink_glass	0	0	0	0	83	1	0	0	0	0	0	0	0	0
Eat_meat	0	0	0	0	7	52	0	0	0	0	0	0	0	0
Eat_soup	0	0	0	0	1	1	8	0	0	0	0	0	0	0
Getup_bed	0	0	0	0	2	0	1	66	0	0	1	0	1	0
Liedown_bed	0	0	0	0	0	0	0	10	0	0	2	0	0	0
Pour_water	0	0	0	0	0	2	0	0	0	46	0	0	0	0
Sitdown_chair	0	0	0	0	1	0	0	13	0	0	17	0	0	0
Standup_chair	0	0	0	0	0	0	0	7	0	0	2	2	0	0
Use_telephone	0	1	0	0	5	0	0	1	0	0	0	0	11	0
Walk	0	32	0	0	0	0	0	1	0	0	1	0	0	56

Segmentation of the vector

```
def segmentData(data, segmentSize):
    segmentSize = segmentSize * 3
    incompleteSegmentData = len(data) % segmentSize
    data = data[:len(data) - incompleteSegmentData]
    dataArr = np.array(data)
    dataArr = np.reshape(dataArr, (-1, segmentSize))
    return dataArr
```

K-means

```
n_clusters = 20
k_means = KMeans(n_clusters=n_clusters,
random_state=0).fit(trainDataSegmented)
```

Generating the histogram

```
def createHistograms(dataSegments, labels, fileIdentifiers,
model, n_clusters):
    unique, segmentCounts = np.unique(fileIdentifiers,
return_counts=True)
    numFiles = len(np.unique(fileIdentifiers))
    features = np.zeros(numFiles * (n_clusters + 1), dtype =
int)
    features = features.reshape(numFiles, (n_clusters + 1))
    prevSegment = 0
    for i in range(len(np.unique(fileIdentifiers))):
        start = prevSegment
        end = prevSegment + segmentCounts[i]
        assignment = vq(dataSegments[start:end],
model.cluster_centers_)
        assignmentArr = np.array(assignment[0])
        feature = np.zeros(n_clusters + 1, dtype = 'int')
        assignmentArr = np.array(assignment[0])
        for j in assignmentArr:
            features[i][j] += 1
        features[i][n_clusters] = labels[start]
        prevSegment = end
    return features
```

Classification

```
def crossValidateAndTrain(allData, allLabels, allFileIds, 
n_clusters):
    dataFoldsIdx = createFolds(allData)
    accuracies = []
    cms = []
    for fold in tqdm(dataFoldsIdx):
        trainingData = allData.take(fold[0], axis=0)
        trainingLabels = allLabels.take(fold[0])
        trainingFileIds = allFileIds.take(fold[0])
        testingData = allData.take(fold[1], axis=0)
        testingLabels = allLabels.take(fold[1])
        testingFileIds = allFileIds.take(fold[1])

        accuracy, matrix = classifyAndReturnResults(n_clusters, 
trainingFileIds, trainingData, trainingLabels, testingFileIds, 
testingData, testingLabels)
        accuracies.append(accuracy)
        cms.append(matrix)
    return accuracies, cms


def classifyAndReturnResults(n_clusters, trainFileIdentifiers, 
trainDataSegmented, trainLabels, testFileIdentifiers, 
testDataSegmented, testLabels):
    k_means_segmented = KMeans(n_clusters=n_clusters, 
random_state=0).fit(trainDataSegmented)
    trainFeatures = createHistograms(trainDataSegmented, 
trainLabels, trainFileIdentifiers, k_means_segmented, 
n_clusters)
    
    k_means = KMeans(n_clusters=n_clusters, 
random_state=0).fit(trainFeatures)
```

```

    trainFeatures = createHistograms(trainDataSegmented,
trainLabels, trainFileIdentifiers, k_means_segmented,
n_clusters)
    testFeatures = createHistograms(testDataSegmented,
testLabels, testFileIdentifiers, k_means_segmented, n_clusters)

    randomForestClassifier =
RandomForestClassifier(max_depth=32, random_state=0,
n_estimators=200)
    randomForestClassifier.fit(trainFeatures[0:, 0:n_clusters],
trainFeatures[:, -1])

    prediction = randomForestClassifier.predict(testFeatures[0:,
0:n_clusters])
    accuracy = accuracy_score(testFeatures[:, -1], prediction)
    accuracy = round((accuracy * 100),2)
    confusionMatrix = confusion_matrix(testFeatures[:, -1],
prediction)
    return accuracy, confusionMatrix
```

Source Code

```
import numpy as np
import pandas as pd
import glob
from tqdm import tqdm
from sklearn.cluster import KMeans
from sklearn.model_selection import KFold
from sklearn.model_selection import GroupKFold
from sklearn.cluster import AgglomerativeClustering
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from scipy.cluster.vq import vq
from scipy.cluster.hierarchy import dendrogram, linkage
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sn
import pandas as pd
import matplotlib.pyplot as plt

get_ipython().run_line_magic('matplotlib', 'inline')
hmpLabels = ['Brush_teeth', 'Climb_stairs', 'Comb_hair',
'Descend_stairs', 'Drink_glass', 'Eat_meat', 'Eat_soup',
'Getup_bed', 'Liedown_bed',
             'Pour_water', 'Sitdown_chair', 'Standup_chair',
'Use_telephone', 'Walk']

hmpData = {}

for label in hmpLabels:
    path = 'data/HMP_Dataset/' + label + '/*.txt'
    files = glob.glob(path)
    hmpData[label] = []
    for file in files:
        fileData = pd.read_csv(file, sep=" ",
header=None).values
        hmpData[label].append(fileData)

def segmentData(data, segmentSize):
    segmentSize = segmentSize * 3
    incompleteSegmentData = len(data) % segmentSize
    data = data[:len(data) - incompleteSegmentData]
    dataArr = np.array(data)
    dataArr = np.reshape(dataArr, (-1, segmentSize))
    return dataArr

def segmentAndSplitData(trainSize, segmentSize, hmpData,
hmpLabels, showOutput = True):
    if (showOutput):
        print ("Using segment size", segmentSize, "and train 
size", trainSize)
        print ('Segmenting classes')
    
    allTrainDataSegmented = []
    allTrainLabels = []
    allTrainFileIdentifiers = []
    allTestDataSegmented = []
    allTestLabels = []
    allTestFileIdentifiers = []
    uniqueTrainFileCtr = 0
    uniqueTestFileCtr = 0
    
    totalFilesAcrossClasses = 0
    totalTrainFilesAcrossClasses = 0
    totalTestFilesAcrossClasses = 0
    
    for hmpClass in hmpLabels:
        totalFiles = len(hmpData[hmpClass])
        trainFiles = int(round(totalFiles ∗ trainSize))
        testFiles = totalFiles − trainFiles
        if (showOutput):
            print (hmpClass, '| Total files:', totalFiles, '|', 
trainFiles, "training |", testFiles, "testing)")
        
        # Calculate totals for files across all classes
        totalFilesAcrossClasses = totalFilesAcrossClasses + 
totalFiles
        totalTrainFilesAcrossClasses = 
totalTrainFilesAcrossClasses + trainFiles
        totalTestFilesAcrossClasses = 
totalTestFilesAcrossClasses + testFiles
        
        # Segment training data
        trainData = []
        for i in range(trainFiles):
            trainData = hmpData[hmpClass][i].flatten()
            trainDataSegmented = segmentData(trainData,
segmentSize)

            for i in range(len(trainDataSegmented)):          #
Assign unique file identifier to each segment

allTrainFileIdentifiers.append(uniqueTrainFileCtr)
                allTrainLabels.append(hmpLabels.index(hmpClass))
            uniqueTrainFileCtr = uniqueTrainFileCtr + 1

            allTrainDataSegmented.extend(trainDataSegmented)

        # Segment test data
        testData = []
        for i in range(trainFiles, totalFiles):
            testData = hmpData[hmpClass][i].flatten()
            testDataSegmented = segmentData(testData,
segmentSize)

            for i in range(len(testDataSegmented)):          #
Assign unique file identifier to each segment
                allTestFileIdentifiers.append(uniqueTestFileCtr)
                allTestLabels.append(hmpLabels.index(hmpClass))
            uniqueTestFileCtr = uniqueTestFileCtr + 1

            allTestDataSegmented.extend(testDataSegmented)

    allTrainDataSegmentedArr = np.array(allTrainDataSegmented)
    allTestDataSegmentedArr = np.array(allTestDataSegmented)
    allTrainFileIdentifiersArr =
np.array(allTrainFileIdentifiers)
    allTrainLabelsArr = np.array(allTrainLabels)
    allTestLabelsArr = np.array(allTestLabels)
    allTestFileIdentifiersArr = np.array(allTestFileIdentifiers)
    if (showOutput):
        print ('Summary for all classes:')
        print (len(allTrainDataSegmentedArr), 'total training
segments |',  len(allTestDataSegmentedArr), 'total test
segments')
        print ('Total files:', totalFilesAcrossClasses, '|',
totalTrainFilesAcrossClasses, "training |",
totalTestFilesAcrossClasses, "testing)")
    return allTrainDataSegmentedArr, allTrainLabelsArr,
allTrainFileIdentifiersArr, allTestDataSegmentedArr,
allTestLabelsArr, allTestFileIdentifiersArr

trainSize = 2/3
segmentSize = 1
(trainDataSegmented, trainLabels, trainFileIdentifiers,
testDataSegmented, testLabels, testFileIdentifiers) =
segmentAndSplitData(trainSize, segmentSize, hmpData, hmpLabels)

allDataSegmented = np.concatenate((trainDataSegmented,
testDataSegmented), axis=0)
allFileIdentifiers = np.concatenate((trainFileIdentifiers,
testFileIdentifiers), axis=None)
allLabels = np.concatenate((trainLabels, testLabels), axis=None)

n_clusters = 20
k_means = KMeans(n_clusters=n_clusters,
random_state=0).fit(trainDataSegmented)

plt.scatter(trainDataSegmented[:,0],trainDataSegmented[:,1],
c=k_means.labels_)

def createHistograms(dataSegments, labels, fileIdentifiers,
model, n_clusters):
    unique, segmentCounts = np.unique(fileIdentifiers,
return_counts=True)
    numFiles = len(np.unique(fileIdentifiers))
    features = np.zeros(numFiles * (n_clusters + 1), dtype =
int)
    features = features.reshape(numFiles, (n_clusters + 1))
    prevSegment = 0
    for i in range(len(np.unique(fileIdentifiers))):
        start = prevSegment
        end = prevSegment + segmentCounts[i]
        assignment = vq(dataSegments[start:end],
model.cluster_centers_)
        assignmentArr = np.array(assignment[0])
        feature = np.zeros(n_clusters + 1, dtype = 'int')
        assignmentArr = np.array(assignment[0])
        for j in assignmentArr:
            features[i][j] += 1
        features[i][n_clusters] = labels[start]
        prevSegment = end
    return features

trainFeatures = createHistograms(trainDataSegmented,
trainLabels, trainFileIdentifiers, k_means, n_clusters)
testFeatures = createHistograms(testDataSegmented, testLabels,
testFileIdentifiers, k_means, n_clusters)

hist = np.zeros(14 * n_clusters, dtype = int)
hist = hist.reshape(14, n_clusters)

for i in range(0, 14):
    hist[i] = trainFeatures[trainFeatures[0:, 14] ==
i].sum(axis=0)[:-1]

_, ax = plt.subplots(nrows=2, ncols=7, figsize=(24,7))

for i in range(0, 7):
    ax[0,i].set_title(hmpLabels[i])
    ax[0,i].hist(hist[i], normed=True, bins=100)

```

```
for i in range(0, 7):
    ax[1,i].set_title(hmpLabels[i+7])
    ax[1,i].hist(hist[i+7], normed=True, bins=100)
```

```
randomForestClassifier = RandomForestClassifier(max_depth=32,
random_state=0, n_estimators=200)
randomForestClassifier.fit(trainFeatures[0:, 0:n_clusters],
trainFeatures[:, -1])
prediction = randomForestClassifier.predict(testFeatures[0:,
0:n_clusters])
accuracy = accuracy_score(testFeatures[:, -1], prediction)
print("Classifier accuracy: " + str(round((accuracy * 100),2)) +
"%")
print(confusion_matrix(testFeatures[:, -1], prediction))
```

```
clustersToTry = [16, 20, 25, 30, 40]
segmentSizesToTry = [1, 4, 8]
inertiasBySegment = {}
accPerClusterNumAndSegmentSize = []
trainSize = 2/3
```

```
for segmentSize in tqdm(segmentSizesToTry):
    inertias = []
    for n_clusters in clustersToTry:
        (trainDataSegmented, trainLabels, trainFileIdentifiers,
testDataSegmented, testLabels, testFileIdentifiers) =
segmentAndSplitData(trainSize, segmentSize, hmpData, hmpLabels,
False)
        k_means_segmented = KMeans(n_clusters=n_clusters,
random_state=0).fit(trainDataSegmented)
        trainFeatures = createHistograms(trainDataSegmented,
trainLabels, trainFileIdentifiers, k_means_segmented,
n_clusters)

        k_means = KMeans(n_clusters=n_clusters,
random_state=0).fit(trainFeatures)
```

```
        inertias.append(k_means.inertia_)

        trainFeatures = createHistograms(trainDataSegmented,
trainLabels, trainFileIdentifiers, k_means_segmented,
n_clusters)
        testFeatures = createHistograms(testDataSegmented,
testLabels, testFileIdentifiers, k_means_segmented, n_clusters)

        # Get accuracy
        randomForestClassifier =
RandomForestClassifier(max_depth = 32, random_state = 0,
n_estimators=10)
        randomForestClassifier.fit(trainFeatures[0:,
0:n_clusters], trainFeatures[:, -1])
        prediction =
randomForestClassifier.predict(testFeatures[0:, 0:n_clusters])
        accuracy = accuracy_score(testFeatures[:, -1],
prediction)
        acc = (accuracy_score(testFeatures[:, -1],
prediction))*100
        acc = round(float(acc), 4)
        accPerClusterNumAndSegmentSize.append([n_clusters,
segmentSize, acc])

    inertiasBySegment[segmentSize] = inertias

accPerClusterNumAndSegmentSize =
np.array(accPerClusterNumAndSegmentSize)
bestAccuracy =
accPerClusterNumAndSegmentSize[accPerClusterNumAndSegmentSize[:,
2].argsort()][-1]
print(f'Best accuracy of {bestAccuracy[2]}% with
{bestAccuracy[0]} clusters and {bestAccuracy[1]} segments')
print(accPerClusterNumAndSegmentSize)

for segmentSize in inertiasBySegment:
    plt.plot(clustersToTry, inertiasBySegment[segmentSize],
marker="o")
    plt.xlabel('Number of clusters')
    plt.ylabel('Average distance to cluster centers')
    plt.title('Elbow plot using segment size of ' +
str(segmentSize))
    plt.grid(True)
    plt.show()

xs = accPerClusterNumAndSegmentSize[:, 0]
ys = accPerClusterNumAndSegmentSize[:, 1]
zs = accPerClusterNumAndSegmentSize[:, 2]

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.scatter(xs, ys, zs, c='r', marker='o')

ax.set_xlabel('Number of clusters')
ax.set_ylabel('Segment size')
ax.set_zlabel('Accuracy')

plt.show()

print(f'Best accuracy of {bestAccuracy[2]}% with
{bestAccuracy[0]} clusters and {bestAccuracy[1]} segments')
segmentSize = 1
n_clusters = 20
k_means = KMeans(n_clusters=n_clusters,
random_state=0).fit(trainDataSegmented)
trainFeatures = createHistograms(trainDataSegmented,
trainLabels, trainFileIdentifiers, k_means, n_clusters)
testFeatures = createHistograms(testDataSegmented, testLabels,
testFileIdentifiers, k_means, n_clusters)
hist = np.zeros(14 * n_clusters, dtype = int)
hist = hist.reshape(14, n_clusters)
```

```
for i in range(0, 14):
    hist[i] = trainFeatures[trainFeatures[0:, 14] == i].sum(axis=0)[:-1]

_, ax = plt.subplots(nrows=2, ncols=7, figsize=(24,7))

for i in range(0, 7):
    ax[0,i].set_title(hmpLabels[i])
    ax[0,i].hist(hist[i], normed=True, bins=100)

for i in range(0, 7):
    ax[1,i].set_title(hmpLabels[i+7])
    ax[1,i].hist(hist[i+7], normed=True, bins=100)

def classifyAndReturnResults(n_clusters, trainFileIdentifiers, trainDataSegmented, trainLabels, testFileIdentifiers, testDataSegmented, testLabels):
    k_means_segmented = KMeans(n_clusters=n_clusters, random_state=0).fit(trainDataSegmented)
    trainFeatures = createHistograms(trainDataSegmented, trainLabels, trainFileIdentifiers, k_means_segmented, n_clusters)

    k_means = KMeans(n_clusters=n_clusters, random_state=0).fit(trainFeatures)

    trainFeatures = createHistograms(trainDataSegmented, trainLabels, trainFileIdentifiers, k_means_segmented, n_clusters)
    testFeatures = createHistograms(testDataSegmented, testLabels, testFileIdentifiers, k_means_segmented, n_clusters)

    randomForestClassifier = RandomForestClassifier(max_depth=32, random_state=0, n_estimators=200)
    randomForestClassifier.fit(trainFeatures[0:, 0:n_clusters], 
trainFeatures[:, -1])
    
    prediction = randomForestClassifier.predict(testFeatures[0:, 
0:n_clusters])
    accuracy = accuracy_score(testFeatures[:, -1], prediction)
    accuracy = round((accuracy * 100),2)
    confusionMatrix = confusion_matrix(testFeatures[:, -1], 
prediction)
    return accuracy, confusionMatrix

def createFolds(data):
    folds = []
    kf = KFold(n_splits=3, shuffle=True)
    for train_indexes, test_indexes in kf.split(data):
        fold = np.array([train_indexes, test_indexes])
        folds.append(fold)
    folds = np.array(folds)
    return folds

def crossValidateAndTrain(allData, allLabels, allFileIds, 
n_clusters):
    dataFoldsIdx = createFolds(allData)
    accuracies = []
    cms = []
    for fold in tqdm(dataFoldsIdx):
        trainingData = allData.take(fold[0], axis=0)
        trainingLabels = allLabels.take(fold[0])
        trainingFileIds = allFileIds.take(fold[0])
        testingData = allData.take(fold[1], axis=0)
        testingLabels = allLabels.take(fold[1])
        testingFileIds = allFileIds.take(fold[1])

        accuracy, matrix = classifyAndReturnResults(n_clusters, 
trainingFileIds, trainingData, trainingLabels, testingFileIds, 
testingData, testingLabels)
```

```
        accuracies.append(accuracy)
        cms.append(matrix)
    return accuracies, cms

trainSize = 2/3
segmentSize = 1
(trainDataSegmented, trainLabels, trainFileIdentifiers,
testDataSegmented, testLabels, testFileIdentifiers) =
segmentAndSplitData(trainSize, segmentSize, hmpData, hmpLabels,
False)

for i in range(len(np.unique(testFileIdentifiers))):
    for j in np.where(testFileIdentifiers == i)[0]:
        testFileIdentifiers[j] = i + 561

allDataSegmented = np.concatenate((trainDataSegmented,
testDataSegmented), axis=0)
allFileIdentifiers = np.concatenate((trainFileIdentifiers,
testFileIdentifiers), axis=None)
allLabels = np.concatenate((trainLabels, testLabels), axis=None)
results = crossValidateAndTrain(allDataSegmented,
allLabels,allFileIdentifiers, 20)
best_accuracy, best_matrix = results[0][np.argmax(results[0])],
results[1][np.argmax(results[0])]

print(f'Best Accuracy of {best_accuracy}%')
print(best_matrix)

df_cm = pd.DataFrame(best_matrix, hmpLabels,hmpLabels)
sn.set(font_scale=1)#for label size
sn.heatmap(df_cm, annot=True,annot_kws={"size": 16})# font size
```