CS498 AML,AMO HW5 Report

Shrashti Singhal, Ankush Singhal

TOTAL POINTS

100 / 100

QUESTION 1

1 Experiment table **40 / 40**

✓ **- 0 pts** **Accuracy greater than 60%**

- **- 2 pts** Maximum Accuracy between 55-60%
- **- 5 pts** Maximum Accuracy between 50%-55%
- **- 10 pts** Maximum Accuracy between 40%-50%
- **- 15 pts** Maximum Accuracy between 30%-40%
- **- 20 pts** Maximum Accuracy between 20%-30%
- **- 25 pts** Maximum Accuracy less than 20%
- **- 40 pts** NA

QUESTION 2

2 Histograms **28 / 28**

✓ **- 0 pts** **14 histograms (2pts per activity)**

QUESTION 3

3 Confusion matrix **22 / 22**

✓ **- 0 pts** **Correct - Diagonal Entries should be large | Possible confusion between "climb stairs-descend stairs", "eat meat-eat soup" (similar pairs)**

- **- 12 pts** Seems incorrect/uninterpretable/confusing
- **- 1 pts** No explicit values in the confusion matrix.
- **- 2 pts** The values in the confusion matrix are larger than the number of data for some categories in one fold. In other words, should't sum the confusion matrix of the three folds. Instead, just present one matrix with the lowest error.

QUESTION 4

4 Code snippets **10 / 10**

✓ **- 0 pts** **Correct**

- **- 3 pts** Segmentation/Window length sample code not available
- **- 2 pts** k-means code not available
- **- 3 pts** conversion to histogram features code not available
- **- 2 pts** classifier training code not available
- **- 2 pts** Snippets are too vague to read. Please contact TAs with clear code snippets for regrading.

QUESTION 5

5 Late **0 / 0**

✓ **- 0 pts** **On time**

- **- 5 pts** 1 day
- **- 10 pts** 2 days
- **- 15 pts** 3 days
- **- 20 pts** 4 days
- **- 25 pts** 5 days
- **- 30 pts** 6 days

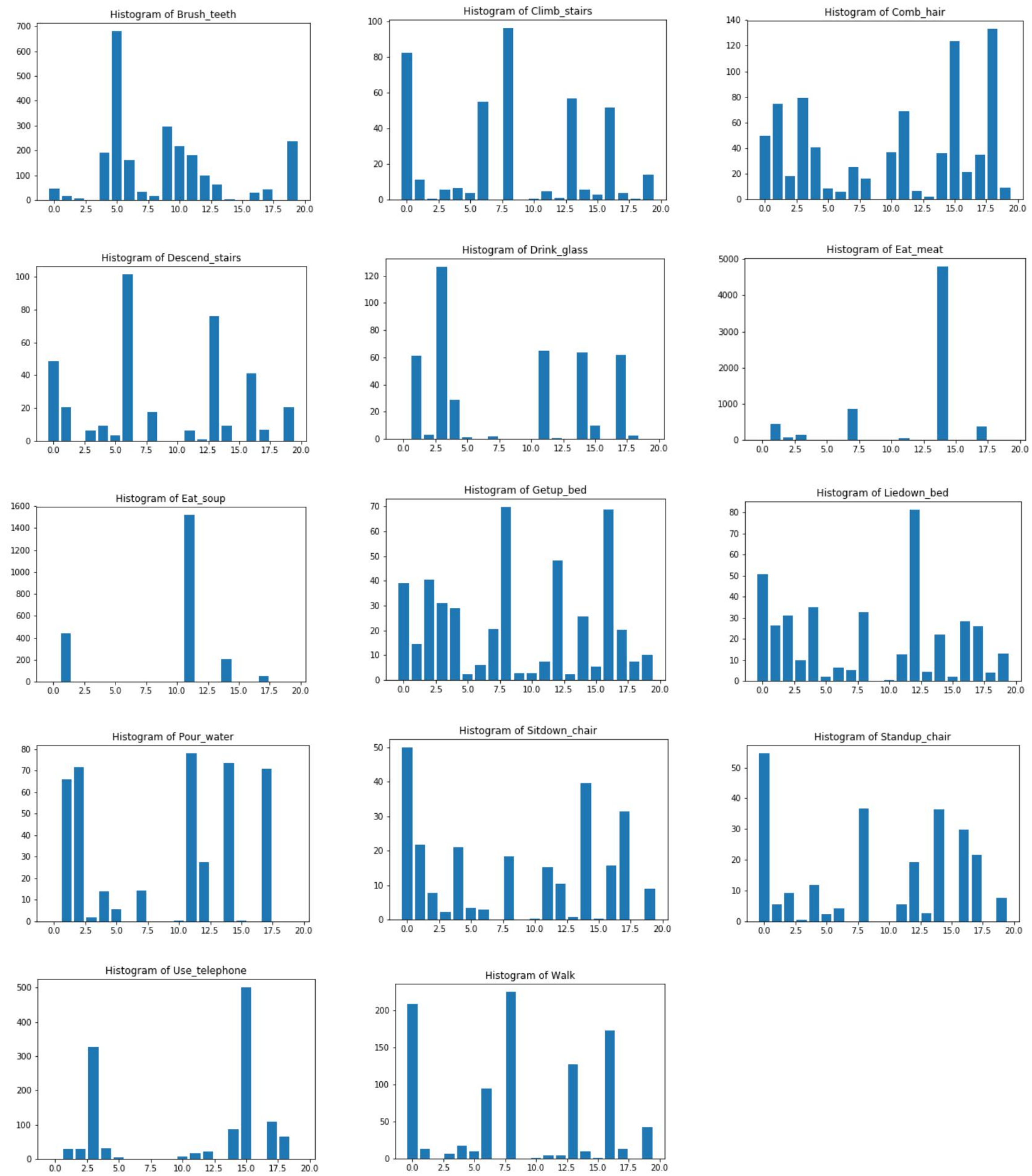
 By: Shrashti Singhal & Ankush Singhal

Part 1 Experiment table:

For K-means please we have used Standard K-means. Following is the output generated using Random Forest Classifier:

```
Segment:  1    Cluster:  4     Accuracy:  71.73913043478261 %
Segment:  1    Cluster:  8     Accuracy:  78.98550724637681 %
Segment:  1    Cluster:  16     Accuracy:  86.59420289855072 %
Segment:  1    Cluster:  20     Accuracy:  87.68115942028986 %
Segment:  1    Cluster:  25     Accuracy:  83.69565217391305 %
Segment:  1    Cluster:  30     Accuracy:  83.69565217391305 %
Segment:  1    Cluster:  40     Accuracy:  84.05797101449275 %
Segment:  1    Cluster:  50     Accuracy:  81.52173913043478 %
Segment:  4    Cluster:  4     Accuracy:  70.28985507246377 %
Segment:  4    Cluster:  8     Accuracy:  78.98550724637681 %
Segment:  4    Cluster:  16     Accuracy:  85.5072463768116 %
Segment:  4    Cluster:  20     Accuracy:  82.6086956521739 %
Segment:  4    Cluster:  25     Accuracy:  86.95652173913044 %
Segment:  4    Cluster:  30     Accuracy:  85.86956521739131 %
Segment:  4    Cluster:  40     Accuracy:  82.6086956521739 %
Segment:  4    Cluster:  50     Accuracy:  83.33333333333334 %
Segment:  8    Cluster:  4     Accuracy:  68.47826086956522 %
Segment:  8    Cluster:  8     Accuracy:  77.17391304347827 %
Segment:  8    Cluster:  16     Accuracy:  78.62318840579711 %
Segment:  8    Cluster:  20     Accuracy:  81.52173913043478 %
Segment:  8    Cluster:  25     Accuracy:  80.07246376811594 %
Segment:  8    Cluster:  30     Accuracy:  81.88405797101449 %
Segment:  8    Cluster:  40     Accuracy:  82.97101449275362 %
Segment:  8    Cluster:  50     Accuracy:  82.2463768115942 %
Segment:  32    Cluster:  4     Accuracy:  66.30434782608695 %
Segment:  32    Cluster:  8     Accuracy:  69.92753623188406 %
Segment:  32    Cluster:  16     Accuracy:  75.36231884057972 %
Segment:  32    Cluster:  20     Accuracy:  73.55072463768117 %
Segment:  32    Cluster:  25     Accuracy:  77.17391304347827 %
Segment:  32    Cluster:  30     Accuracy:  72.46376811594203 %
Segment:  32    Cluster:  40     Accuracy:  73.55072463768117 %
Segment:  32    Cluster:  50     Accuracy:  74.27536231884058 %
Segment:  64    Cluster:  4     Accuracy:  61.23188405797102 %
Segment:  64    Cluster:  8     Accuracy:  71.73913043478261 %
Segment:  64    Cluster:  16     Accuracy:  72.46376811594203 %
Segment:  64    Cluster:  20     Accuracy:  73.18840579710145 %
Segment:  64    Cluster:  25     Accuracy:  73.91304347826086 %
Segment:  64    Cluster:  30     Accuracy:  69.56521739130434 %
Segment:  64    Cluster:  40     Accuracy:  71.01449275362319 %
Segment:  64    Cluster:  50     Accuracy:  68.11594202898551 %
Best accuracy of 87.68115942028986% with 20.0 clusters and 1.0 segments
```

Part 2 Histograms: Used the Best accuracy clusters (20) and Segment (1) value for draw Histogram



Part 3 Confusion matrix:

0: Sitdown_chair, 1: Eat_meat, 2: Walk, 3: Use_telephone, 4: Drink_glass, 5: Descend_stairs, 6: Standup_chair, 7: Eat_soup, 8: Getup_bed, 9: Climb_stairs, 10: Brush_teeth, 11: Pour_water, 12: Liedown_bed, 13: Comb_hair

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	3	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	33	0	1	0	0	0	0	0	0	0	0	0	0
2	0	0	8	0	0	0	0	1	0	0	0	0	1	0
3	0	2	0	12	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	33	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	28	0	0	0	5	0	0
8	0	0	0	0	0	0	0	3	2	0	2	2	0	0
9	0	0	0	0	0	0	0	0	0	33	0	0	0	0
10	0	0	0	0	0	0	0	1	1	0	24	7	0	0
11	0	0	0	0	0	0	0	1	0	0	9	24	0	0
12	0	0	0	0	1	0	0	0	0	0	0	0	3	0
13	0	7	0	0	0	0	0	1	0	0	0	0	0	25

Part 4 A screenshot of your code:

i) Segmentation of the vector

```
for j in range(train_num):
    train_activity = []
    cur_file = act_data[i][j]
    length = cur_file.shape[0]
    segment_num = math.floor(length/segment_size)

    for k in range(segment_num):
        train_activity.append(cur_file[k*segment_size:(k+1)*segment_size].T.flatten()[:segment_size*3+1])
        activities.append(cur_file[k*segment_size:(k+1)*segment_size].T.flatten()[:segment_size*3+1])
    train_activity = np.array(train_activity)
    train_activities.append(train_activity)

for j in range(train_num, train_num+test_num):
    test_activity = []
    cur_file = act_data[i][j]
    length = cur_file.shape[0]
    segment_num = math.floor(length/segment_size)

    for k in range(segment_num):
        test_activity.append(cur_file[k*segment_size:(k+1)*segment_size].T.flatten()[:segment_size*3+1])

    test_activity = np.array(test_activity)
    test_activities.append(test_activity)
```

ii) K-means

```
def KMeans(activities, cluster_size, segment_size):
    kmeans = KMeans(n_clusters=cluster_size, random_state=0).fit(activities[:, :segment_size*3])
    train_centers = kmeans.cluster_centers_
    train_labels = kmeans.labels_
    return kmeans, train_labels, train_centers

def KMeansPredict(model, data):
    return model.predict(data)
```

```
model, train_labels, train_centers = KMeans(activities, cluster_size, segment_size)
```

```
for j in range(act_test[i].shape[0]):           # each segment
    test_label.append(KMeansPredict(model, act_test[i][j, :segment_size*3].reshape(1, -1)))
```

iii) Generating the histogram

```
def draw(histograms, signals, cluster_size, act_name):
    for i in range(14):
        histogram_sum, count = np.zeros(cluster_size), 0.0
        for (histogram, signal) in zip(histograms, signals):
            if signal == i:
                histogram_sum += histogram
                count += 1.0
        histogram_sum /= count
        plt.bar(range(cluster_size), histogram_sum)
        plt.title('Histogram of ' + act_name[i])
        plt.savefig('%s.png'%i)
        plt.close()
```

iv) Classification

```
rf = RF(max_depth=32, random_state=0, n_estimators=200).fit(train_histogram, train_signal)
accurate = 0
cov_matrix = dict()
for i in range(test_samples):
    label = rf.predict(test_histogram[i].reshape(1, -1))[0]
    label_ori = act_test[i][0, segment_size*3]
    if label_ori not in cov_matrix:
        cov_matrix[label_ori] = [0]*14
    cov_matrix[label_ori][label] += 1
    if int(label) == label_ori:
        accurate = accurate + 1
```

Part 5 Screenshots of all your source code:

```python
import os
import math
import random
import numpy as np
import pandas as pd
from collections import defaultdict
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier as RF

act_num = 14

def readData(path):
    # activities: data of all activities
    # activity: data of each activity
    # cur_file: data of each file
    # cur_act: data of each line
    folders = os.listdir(path)
    folders = [x for x in folders if 'MODEL' not in x and 'DS_Store' not in x]

    #print(folders)

    activities = []
    for i in range(act_num):
        activity = []
        path1 = "" + path + "/" + folders[i]
        files = os.listdir(path1)
        random_select = random.sample(range(len(files)), len(files))
        for j in random_select:
            file = files[j]
            cur_file = []
            if not os.path.isdir(file):
                with open("" + path1 + "/" + file, 'r') as f:
                    for line in f.readlines():
                        cur_act = []
                        num = str(line).rstrip("\r\n").split(" ")
                        cur_act.append(int(num[0]))
                        cur_act.append(int(num[1]))
                        cur_act.append(int(num[2]))
                        cur_act.append(i)
                        cur_act = np.array(cur_act)
                        cur_file.append(cur_act)
                    cur_file = np.array(cur_file)
                    activity.append(cur_file)
        activity = np.array(activity)
        activities.append(activity)

    activities = np.array(activities)
    #print(activities)
    return folders, activities
```

```python
def splitData(act_data, percent, segment_size):
    activities = []
    train_activities = []
    test_activities = []

    for i in range(act_num):
        file_num = act_data[i].shape[0]
        test_num = math.floor(file_num*(1-percent))
        if(test_num < 1):
            test_num = 1
        train_num = file_num-test_num

        for j in range(train_num):
            train_activity = []
            cur_file = act_data[i][j]
            length = cur_file.shape[0]
            segment_num = math.floor(length/segment_size)

            for k in range(segment_num):
                train_activity.append(cur_file[k*segment_size:(k+1)*segment_size].T.flatten()[:segment_size*3+1])
                activities.append(cur_file[k*segment_size:(k+1)*segment_size].T.flatten()[:segment_size*3+1])
            train_activity = np.array(train_activity)
            train_activities.append(train_activity)

        for j in range(train_num, train_num+test_num):
            test_activity = []
            cur_file = act_data[i][j]
            length = cur_file.shape[0]
            segment_num = math.floor(length/segment_size)

            for k in range(segment_num):
                test_activity.append(cur_file[k*segment_size:(k+1)*segment_size].T.flatten()[:segment_size*3+1])

            test_activity = np.array(test_activity)
            test_activities.append(test_activity)
    activities = np.array(activities)
    train_activities = np.array(train_activities)
    test_activities = np.array(test_activities)

    return activities,  train_activities, test_activities

def TrainHistogram(data, cluster_size, labels, segment_size):
    length = data.shape[0]
    index = 0
    label = []
    histograms = []
    signals = []
    for i in range(length):
        signal = data[i][0, segment_size*3]
        signals.append(signal)
        count = np.zeros(cluster_size)
        for j in range(data[i].shape[0]):
            label = labels[index]
            count[label] = count[label] + 1
```

```
            index += 1
        histograms.append(count)
    histograms = np.array(histograms)
    signals = np.array(signals)

    for i in range(length):
        total = np.sum(histograms[i])
        for j in range(cluster_size):
            histograms[i][j] = float(histograms[i][j]) # normalization

    return histograms, signals


def TestHistogram(data, cluster_size, labels):
    count = np.zeros(cluster_size)
    length = data.shape[0]

    for i in range(length):
        label = labels[i]
        count[label] = count[label] + 1

    total = np.sum(count)
    for i in range(cluster_size):
        count[i] = float(count[i])

    return count

def KMeans(activities, cluster_size, segment_size):
    kmeans = KMeans(n_clusters=cluster_size, random_state=0).fit(activities[:, :segment_size*3])
    train_centers = kmeans.cluster_centers_
    train_labels = kmeans.labels_
    return kmeans, train_labels, train_centers

def KMeansPredict(model, data):
    return model.predict(data)

def getCenter(data ,labels, cluster_size, segment_size):
    centers = np.array([np.zeros(segment_size*3)]*cluster_size)
    label_count = defaultdict(float)
    for i in range(data.shape[0]):
        label, point = labels[i], data[i]
        centers[label] += point
        label_count[label] += 1
    for i in range(cluster_size):
        centers[i] = centers[i] / label_count[i]
    return centers

def getLable(data, centers):
    dist, label = float('inf'), 0
    for i in range(centers.shape[0]):
        cur_dist = np.linalg.norm(data-centers[i])
        if cur_dist < dist:
            dist, label = cur_dist ,i
    return np.array([label])
```

```
def draw(histograms, signals, cluster_size, act_name):
    for i in range(14):
        histogram_sum, count = np.zeros(cluster_size), 0.0
        for (histogram, signal) in zip(histograms, signals):
            if signal == i:
                histogram_sum += histogram
                count += 1.0
        histogram_sum /= count
        plt.bar(range(cluster_size), histogram_sum)
        plt.title('Histogram of ' + act_name[i])
        plt.savefig('%s.png'%i)
        plt.close()
```

```python
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, activities, cluster_size, segment_size,act_name):
    dataFoldsIdx = createFolds(allData)
    accuracies = []
    cms = []
    for fold in dataFoldsIdx:
        # Get Data
        act_train = allData.take(fold[0], axis=0)
        act_train = np.array(act_train)
        activities = np.array(activities)
        act_test = allData.take(fold[1], axis=0)
        act_test = np.array(act_test)
        # Train
        model, train_labels, train_centers = KMeans(activities, cluster_size, segment_size)
        #print(model)
        train_histogram, train_signal = TrainHistogram(act_train, cluster_size, train_labels, segment_size)
        draw(train_histogram, train_signal, cluster_size, act_name)
        test_labels = []
        test_samples = act_test.shape[0]
        for i in range(test_samples):                          # each file
            test_label = []
            for j in range(act_test[i].shape[0]):              # each segment
                test_label.append(KMeansPredict(model, act_test[i][j, :segment_size*3].reshape(1, -1)))

            test_label = np.array(test_label)
            test_labels.append(test_label)
        test_labels = np.array(test_labels)
        test_histogram = []
        for i in range(test_samples):
            test_histogram.append(TestHistogram(act_test[i], cluster_size, test_labels[i]))

        rf = RF(max_depth=32, random_state=0, n_estimators=200).fit(train_histogram, train_signal)

        accurate = 0
        cov_matrix = dict()
        for i in range(test_samples):
            label = rf.predict(test_histogram[i].reshape(1, -1))[0]
            label_ori = act_test[i][0, segment_size*3]
            if label_ori not in cov_matrix:
                cov_matrix[label_ori] = [0]*14
            cov_matrix[label_ori][label] += 1
            if int(label) == label_ori:
                accurate = accurate + 1
        cov_df = pd.DataFrame.from_dict(cov_matrix, orient='index', columns=[str(x) for x in range(14)])
        accuracy = (accurate/ len(act_test))*100
        print(accuracy)
        accuracies.append(accuracy)
        cms.append(cov_df)

    return accuracies, cms
```

```
def main():
    act_name, act_data = readData('./HMP_Dataset')

    # Experiments

    print("Experiments...")
    trainSize = 2/3

    clustersToTry = [4, 8, 16, 20, 25, 30, 40, 50]
    segmentSizesToTry = [1, 4, 8, 32, 64]
    inertiasBySegment = {}
    accPerClusterNumAndSegmentSize = []

    for segment_size in segmentSizesToTry:
        inertias = []
        for cluster_size in clustersToTry:
            activities, act_train, act_test = splitData(act_data, trainSize, segment_size)

            model, train_labels, train_centers = KMeans(activities, cluster_size, segment_size)

            inertias.append(model.inertia_)

            train_histogram, train_signal = TrainHistogram(act_train, cluster_size, train_labels, segment_size)
            #draw(train_histogram, train_signal, cluster_size, act_name)

            test_labels = []
            test_samples = act_test.shape[0]
            for i in range(test_samples):                      # each file
                test_label = []
                for j in range(act_test[i].shape[0]):          # each segment
                    test_label.append(kmsPredict(model, act_test[i][j, :segment_size*3].reshape(1, -1)))
                    # test_label.append(getLable((act_test[i][j, :segment_size*3]), train_centers))
                test_label = np.array(test_label)
                test_labels.append(test_label)
            test_labels = np.array(test_labels)

            test_histogram = []
            for i in range(test_samples):
                test_histogram.append(TestHistogram(act_test[i], cluster_size, test_labels[i]))

            rf = RF(max_depth=32, random_state=0, n_estimators=200).fit(train_histogram, train_signal)
            accurate = 0
            cov_matrix = dict()
            for i in range(test_samples):
                label = rf.predict(test_histogram[i].reshape(1, -1))[0]
                label_ori = act_test[i][0, segment_size*3]
                if label_ori not in cov_matrix:
                    cov_matrix[label_ori] = [0]*14
                cov_matrix[label_ori][label] += 1
                if int(label) == label_ori:
                    accurate = accurate + 1
            #cov_df = pd.DataFrame.from_dict(cov_matrix, orient='index', columns=[str(x) for x in range(14)])
            #cov_df.to_csv('cov.csv',index=False)
            acc = (accurate/ len(act_test))*100
            print(" ")
            print('Segment: ',segment_size,'   Cluster: ', cluster_size,'    Accuracy: ', acc,'%' )
            #print(cov_df)
            accPerClusterNumAndSegmentSize.append([cluster_size, segment_size, acc])

        inertiasBySegment[segment_size] = inertias

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

    segment_size= int(bestAccuracy[1])
    cluster_size= int(bestAccuracy[0])
    matrix_output=True

    activities, act_train, act_test = splitData(act_data, trainSize, segment_size)

    allDataSegmented = np.concatenate((act_train, act_test), axis=0)
    allDataSegmented = np.array(allDataSegmented)
    #print(allDataSegmented)

    results = crossValidateAndTrain(allDataSegmented, activities, cluster_size, segment_size,act_name )

    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)

main()
```