1. Page 1 (40 pts) Experiment table

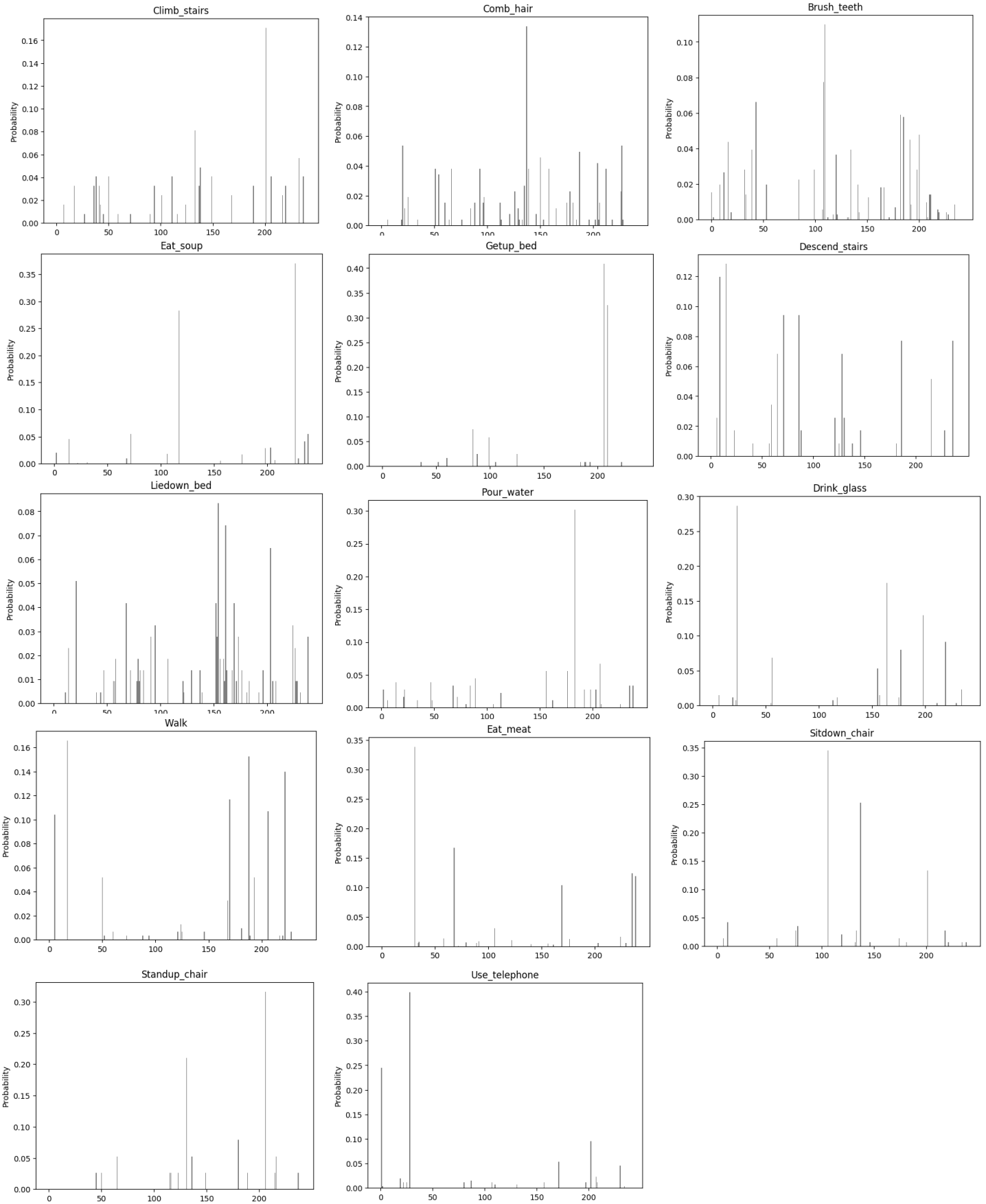
Table listing the experiments carried out with the following columns. Size of the fixed length sample Overlap (0-X%) K-value Classifier Accuracy. We expect you to have tried at least 2 values of K and at least 2 different lengths of the windows for quantization. Note: For K-means please also list if you used standard K-means or hierarchical.

Size of the fixed length sample	Overlap	K-value	Classifier	Accuracy
24	0.9	120	Random forest	0.796491228
32	0.9	120	Random forest	0.821052632
40	0.9	120	Random forest	0.838596491
24	0.5	120	Random forest	0.78245614
32	0.5	120	Random forest	0.796491228
40	0.5	120	Random forest	0.785964912
24	0	120	Random forest	0.712280702
32	0	120	Random forest	0.754385965
40	0	120	Random forest	0.733333333
24	0.9	240	Random forest	0.828070175
32	0.9	240	Random forest	0.838596491
40	0.9	240	Random forest	0.807017544
24	0.5	240	Random forest	0.807017544
32	0.5	240	Random forest	0.78245614
40	0.5	240	Random forest	0.78245614
24	0	240	Random forest	0.740350877
32	0	240	Random forest	0.722807018
40	0	240	Random forest	0.715789474
24	0.9	480	Random forest	0.828070175
32	0.9	480	Random forest	0.81754386
40	0.9	480	Random forest	0.821052632
24	0.5	480	Random forest	0.807017544
32	0.5	480	Random forest	0.78245614
40	0.5	480	Random forest	0.771929825
24	0	480	Random forest	0.729824561
32	0	480	Random forest	0.736842105
40	0	480	Random forest	0.712280702

The standard K-means is used for the experiments.

2. Page 2 (28 pts) Histograms

As indicated by the experiment table in Page 1, when using **K = 240, overlap = 0.9, and 32 samples per segment**, the highest accuracy of 83.86% can be achieved. The histogram of each activity has been listed below:



3. Page 3 (22 pts) Confusion matrix

Class confusion matrix from the classifier that you used. Please make sure to label the row/columns of the matrix so that we know which row corresponds to what.

In the confusion matrix below, predictions are in columns and actual values in rows

	Liedown_bed	Climb_stairs	Getup_bed	Comb_hair	Eat_soup	Walk	Drink_glass	Pour_water	Brush_teeth	Sitdown_chair	Descend_stair	Standup_chai	Use_telephone	Eat_meat
Liedown_bed	0	0	5	0	0	0	0	0	0	5	0	0	0	0
Climb_stairs	0	30	0	0	0	2	1	0	0	0	0	1	0	0
Getup_bed	0	0	27	0	0	0	0	0	0	1	0	6	0	0
Comb_hair	0	0	0	11	0	0	0	0	0	0	0	0	0	0
Eat_soup	0	0	0	0	0	0	0	1	0	0	0	0	0	0
Walk	0	6	1	0	0	25	0	0	0	0	0	2	0	0
Drink_glass	0	0	0	0	0	0	33	1	0	0	0	0	0	0
Pour_water	0	0	0	0	0	0	0	32	0	0	0	2	0	0
Brush_teeth	0	0	0	0	0	0	0	0	4	0	0	0	0	0
Sitdown_chair	0	0	0	0	0	0	0	2	0	25	0	7	0	0
Descend_stairs	0	3	1	0	0	2	0	0	0	0	8	0	0	0
Standup_chair	0	0	2	0	0	0	0	0	0	1	0	31	0	0
Use_telephone	0	0	0	0	0	0	1	1	0	0	0	1	2	0
Eat_meat	0	0	0	0	0	0	0	0	0	0	0	0	0	2

4. Page 4 (10 pts) A screenshot of your code

The page should contain snippets of code demonstrating:

i) Segmentation of the vector

```
def generate_segments(data, n_cluster, n_sample, overLap):
    no_of_rows = np.shape(data)[0]
    no_of_col = np.shape(data)[1]
    no_overlap = int(n_sample * overLap)
    data_overlap = np.zeros(no_of_col * n_sample).reshape(1, -1)
    ind = 0
    while ind + n_sample < no_of_rows:
        data_vector = data[ind:ind + n_sample, :].reshape(1, -1)
        data_overlap = np.concatenate((data_overlap, data_vector), axis=0)
        ind += no_overlap
    return data_overlap[1:]
```

ii) K-means

```
def generate_features(data, n_cluster, n_sample, dataPath, ratio, overlap):
    # computing K-Means with K = n_cluster
    centroids, _ = kmeans(data, n_cluster)
    features_train = []
    features_test = []
    for subdir, dirs, files in os.walk(dataPath):
        classes = dirs
        ind = 0
        for folder in dirs:
            dirs = os.listdir(os.path.join(dataPath, folder))
            numFiles = len(dirs)
            for i in range(0, int(numFiles * ratio)):
                train_data = readFiles(os.path.join(dataPath, folder, dirs[i]), n_cluster, n_sample,
overlap)
                # assign each sample to a cluster
                clx_train, _ = vq(train_data, centroids)
                # calculate histograms of cluster centers
                feature = get_histograms(clx_train, n_cluster)
                # store feature values
                feature.append(ind)
                features_train.append(feature)
            ind += 1
    return np.array(features_train), np.array(features_test), classes
```

iii) Generating the histogram

```
                # Plot histogram
                plt.bar(np.arange(len(feature)), feature, align='center', alpha=0.5, color='k')
                plt.xticks([0, 50, 100, 150, 200])
                plt.ylabel('Probability')
                plt.title(classes[ind])
                plt.savefig('histogram/' + classes[ind] + '.png', bbox_inches='tight')  # Save the plot
to a file
                plt.close()  # Close the plot instance to clear the canvas
```

iv) Classification

```
# classify using random forest classifier
clf = RandomForestClassifier(n_estimators=90, max_depth=32, random_state=8)
clf.fit(features_train[:, :n_cluster], features_train[:, n_cluster])
prediction = clf.predict(features_test[:, :n_cluster])
```

5. Page 5+ Screenshots of all your souce code

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import linalg as LA
from sklearn.decomposition import PCA
from scipy.cluster.vq import kmeans, vq
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import os
def get_accuracy(prediction, actual):
    return float(np.sum(prediction == actual) / prediction.shape[0])

def get_histograms(clx, n_cluster):
    feature = []
    unique, counts = np.unique(clx, return_counts=True)
    dictionary = dict(zip(unique, counts))
    for i in range(n_cluster):
        if i in dictionary.keys():
            feature.append(float(dictionary[i] / clx.shape[0]))
        else:
            feature.append(0)
    return feature

def readFiles(dirPath, n_cluster, n_sample, overlap):
    file_data = np.genfromtxt(dirPath, delimiter=' ', dtype=float)
    data = generate_segments(file_data, n_cluster, n_sample, overlap)
    return data

def generate_features(data, n_cluster, n_sample, dataPath, ratio, overlap):
    # computing K-Means with K = n_cluster
    centroids, _ = kmeans(data, n_cluster)
    features_train = []
    features_test = []
    for subdir, dirs, files in os.walk(dataPath):
        classes = dirs
        ind = 0
        for folder in dirs:
            dirs = os.listdir(os.path.join(dataPath, folder))
            numFiles = len(dirs)
            for i in range(0, int(numFiles * ratio)):
                train_data = readFiles(os.path.join(dataPath, folder, dirs[i]), n_cluster, n_sample,
overlap)
                # assign each sample to a cluster
                clx_train, _ = vq(train_data, centroids)
                # calculate histograms of cluster centers
                feature = get_histograms(clx_train, n_cluster)
                # Plot histogram
                plt.bar(np.arange(len(feature)), feature, align='center', alpha=0.5, color='k')
                plt.xticks([0, 50, 100, 150, 200])
                plt.ylabel('Probability')
                plt.title(classes[ind])
                plt.savefig('histogram/' + classes[ind] + '.png', bbox_inches='tight')  # Save the plot
to a file
                plt.close()  # Close the plot instance to clear the canvas
                # store feature values
                feature.append(ind)
                features_train.append(feature)
            for j in range(int(numFiles * ratio), numFiles):
                test_data = readFiles(os.path.join(dataPath, folder, dirs[j]), n_cluster, n_sample,
overlap)
                # assign each sample to a cluster
                clx_test, _ = vq(test_data, centroids)
                # calculate histograms of cluster centers
```

```
                feature = get_histograms(clx_test, n_cluster)
                feature.append(ind)
                features_test.append(feature)
            ind += 1
    return np.array(features_train), np.array(features_test), classes
def generate_segments(data, n_cluster, n_sample, overLap):
    no_of_rows = np.shape(data)[0]
    no_of_col = np.shape(data)[1]
    no_overlap = int(n_sample * overLap)
    data_overlap = np.zeros(no_of_col * n_sample).reshape(1, -1)
    ind = 0
    while ind + n_sample < no_of_rows:
        data_vector = data[ind:ind + n_sample, :].reshape(1, -1)
        data_overlap = np.concatenate((data_overlap, data_vector), axis=0)
        ind += no_overlap
    return data_overlap[1:]

def readFolder(folder, n_cluster, n_sample, ratio, overlap):
    dirs = os.listdir(folder)
    numFiles = len(dirs)
    # Read in training data
    train_data = np.zeros(3).reshape(1, 3)
    for i in range(0, int(numFiles * ratio)):
        file_data = np.genfromtxt(os.path.join(folder, dirs[i]), delimiter=' ', dtype=float)
        train_data = np.concatenate((np.array(train_data), file_data), axis=0)
    train_data = train_data[1:]
    train_data = generate_segments(train_data, n_cluster, n_sample, overlap)
    # Read in testing data
    test_data = np.zeros(3).reshape(1, 3)
    if ratio != 1:
        for i in range(int(numFiles * ratio), numFiles):
            file_data = np.genfromtxt(os.path.join(folder, dirs[i]), delimiter=' ', dtype=float)
            test_data = np.concatenate((test_data, file_data), axis=0)
        test_data = test_data[1:]
        test_data = generate_segments(test_data, n_cluster, n_sample, overlap)
    return train_data, test_data


n_cluster_list = [120, 240, 480] # set the no of cluster
overlap_list = [0.1, 0.5, 1]
n_sample_list = [24, 32, 40] # set the no of samples per segment
# for n_cluster in n_cluster_list:
#     for overlap in overlap_list:
#         for n_sample in n_sample_list:
# ------ Given values -----
n_cluster = 240
overlap = 0.1
n_sample = 32
# ------ Given values -----
#
ratio = float(2 / 3)
dataPath = "../HMP_Dataset"
dirs = os.listdir(dataPath)
train_vectors = np.zeros(n_sample * 3).reshape(1, n_sample * 3)
test_vectors = np.zeros(n_sample * 3).reshape(1, n_sample * 3)
for subdir, dirs, files in os.walk(dataPath):
    for folder in dirs:
        train_data, test_data = readFolder(os.path.join(dataPath, folder), n_cluster, n_sample, 1,
overlap)
        train_vectors = np.concatenate((train_vectors, train_data), axis=0)

# Make features by using K means
features_train, features_test, classes = generate_features(train_vectors[1:], n_cluster, n_sample,
dataPath, ratio, overlap)

# classify using random forest classifier
clf = RandomForestClassifier(n_estimators=90, max_depth=32, random_state=8)
```

```
clf.fit(features_train[:, :n_cluster], features_train[:, n_cluster])
prediction = clf.predict(features_test[:, :n_cluster])
acc = get_accuracy(np.array(prediction), features_test[:, n_cluster])
print("The accuracy is {}% when using K: {}, overlap: {}, n_sample: {}.".format(acc * 100, n_cluster,
overlap, n_sample))



print(confusion_matrix(features_test[:, n_cluster], prediction))
```