Homework 5

1. Table listing the experiments carried out with the following columns.

Size of the Fixed Length Sample	Overlap(0-X%)	K-value	Classifier	Accuracy
32	0	24	SVM	75.78%
20	0	24	SVM	69.57%
16	0	24	SVM	68.32%

Size of the Fixed Length Sample	Overlap(0-X%)	K-value	Classifier	Accuracy
32	0	24	SVM	75.78%
32	0	20	SVM	73.29%
32	0	16	SVM	72.05%

Size of the Fixed Length Sample	Overlap(0-X%)	K-value	Classifier	Accuracy
32	0	24	Random Forest	65.22%
20	0	24	Random Forest	64.60%
16	0	24	Random Forest	62.11%

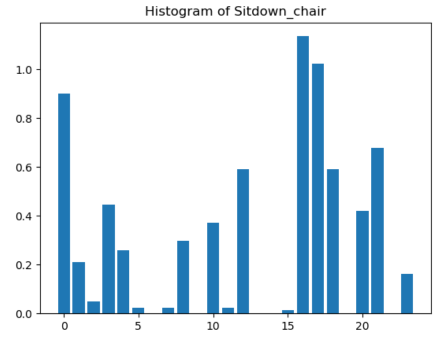
Size of the Fixed Length Sample	Overlap(0-X%)	K-value	Classifier	Accuracy
32	0	24	Random Forest	65.22%
32	0	20	Random Forest	61.49%
32	0	16	Random Forest	66.46%

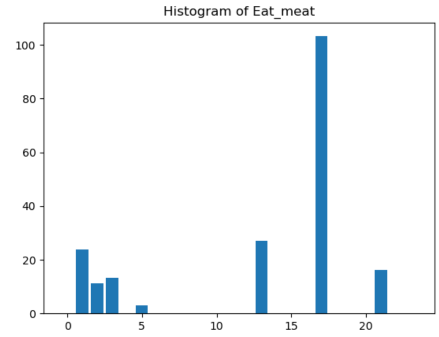
1) All K-means use standard K-means.
2) Test-train split percent is 0.8. 80% of files are training set and 20% of files are testing set. We first scan each folder which include files of each activity and there are 14 folders. For each folder(activity), we randomly choose 80% of file as training set and the left 20% as testing files without overlapping. In each file, we first create segments according to the size of the fixed length sample and reshape each segment's data as [1, size*3+1]. The last data is the original label of the activity.

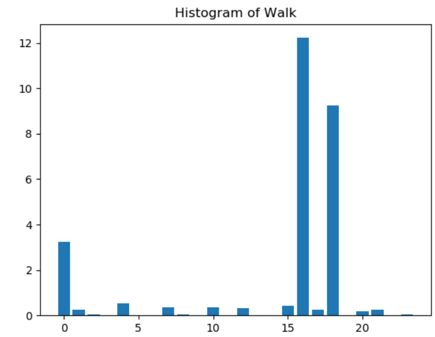
2.

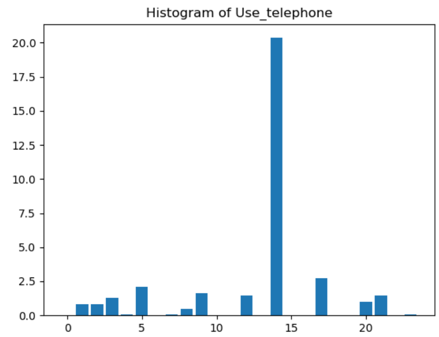
1) Histograms of the mean quantized vector:

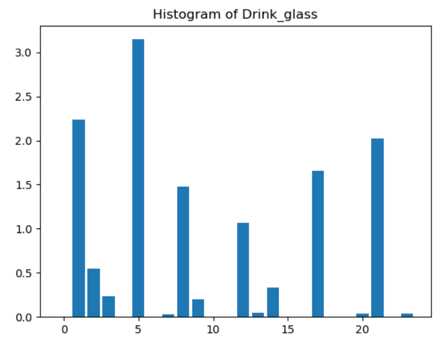
Size of the Fixed Length Sample	Overlap(0-X%)	K-value	Classifier	Accuracy
32	0	24	SVM	75.78%

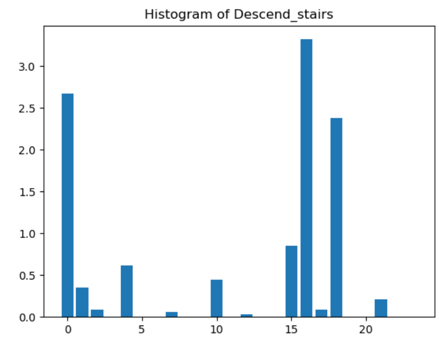


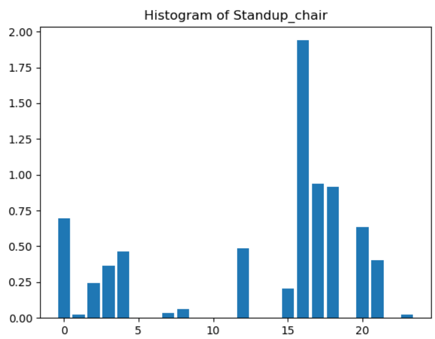


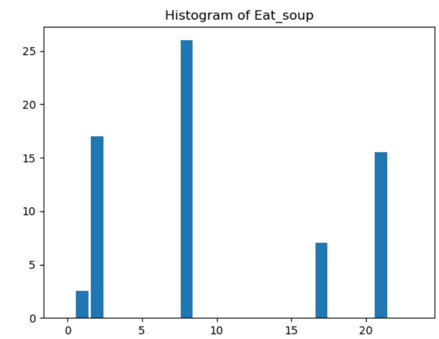


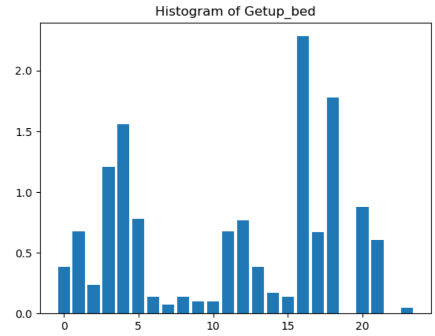


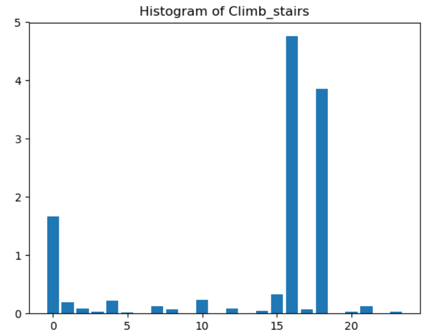


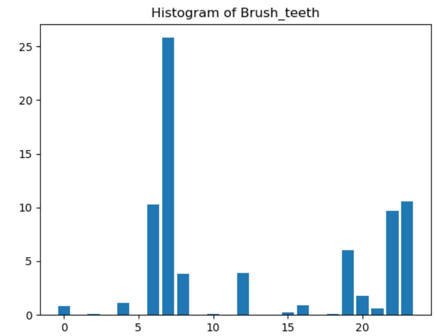


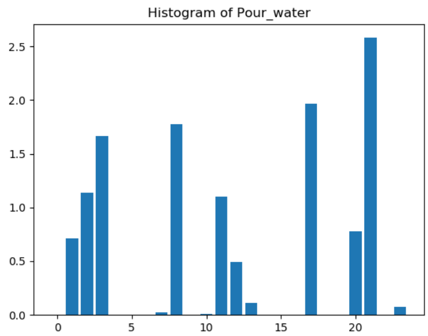


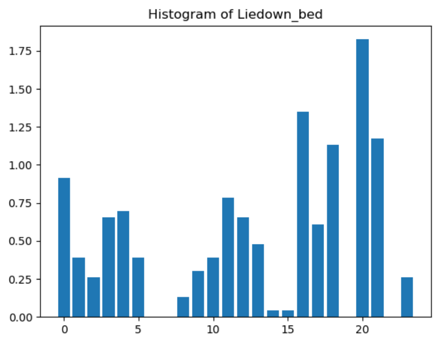


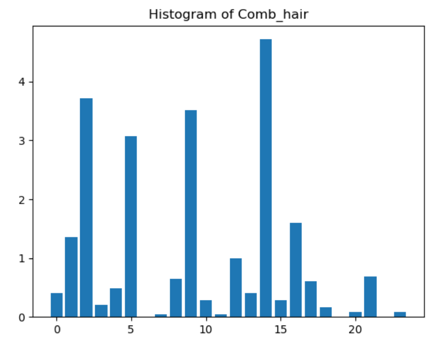












2) Class confusion matrix from the classifier that you used.

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	14	0	0	0	0	0	1	0	4	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	19	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	0	0	0	1
4	0	0	1	0	17	0	0	0	0	0	0	1	0	0
5	0	0	0	0	0	1	1	0	0	6	0	0	0	0
6	2	0	0	0	0	0	16	0	1	0	0	1	0	0
7	0	0	1	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	2	0	17	0	0	0	1	0
9	1	0	1	0	0	0	2	0	0	16	0	0	0	0
10	0	0	2	0	0	0	0	0	0	0	0	0	0	0
11	0	0	2	0	0	0	0	0	0	0	0	17	0	0
12	2	0	0	0	0	0	0	0	1	0	0	1	1	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	6

3. Code snippets highlighting the following:

1) Segmentation of the vector

```
for j in range(train_num):                        # traverse each train file
    train_activity = []
    cur_file = act_data[i][j]                     # current file's data
    length = cur_file.shape[0]                    # number of samples in current file
    segment_num = math.floor(length/segment_size)          # number of segment

    for k in range(segment_num):                  # build up segment
        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):  # traverse each train file
    test_activity = []
    cur_file = act_data[i][j]                     # current file's data
    length = cur_file.shape[0]                    # number of samples in current file
    segment_num = math.floor(length/segment_size)          # number of segment

    for k in range(segment_num):                  # build up segment
        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)
```

2) K-means

```
def kms(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 kmsPredict(model, data):
    return model.predict(data)
```

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

```
 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)))
```

3) 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()
```

4) Classification

```
# rf = RF(max_depth=5, random_state=0).fit(train_histogram, train_signal)
svm = SVC(gamma='auto').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 = svm.predict(test_histogram[i].reshape(1, -1))[0]
    label_ori = act_test[i][0, segment_size*3]
    # print(type(label),label,type(label_ori))
    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
```

4. Analysis:

Since each activity's time/length is different. Thus, we use fixed size of creating multiple segments of each activity. Then, we use k-means to reduce the dimension and create a new feature vector. Though the number of segments at each center is different, the distribution is similar between the same activities and different between different activities. Depending on this, we separate testing data into multiple segments and then find its pattern. Finally, we can calculate the distance between testing pattern and training pattern. We classify the testing activity according to the minimum distance.

According to the confusion matrix, each row means one activity in the original label and each col means the predict label. We can find out the value of the diagonal line is largest which means the correct prediction. Those values which are not on a diagonal line mean incorrect predict the result.

Code Screenshot:

```
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, AgglomerativeClustering
from sklearn.ensemble import RandomForestClassifier as RF
from sklearn.svm import SVC


act_num = 14

def readData(path1):
    # 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(path1)
    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 = []
        path2 = "" + path1 + "/" + folders[i]
        files = os.listdir(path2)
        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("" + path2 + "/" + 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)
    return folders, activities


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

    for i in range(act_num): # each activity
        file_num = act_data[i].shape[0]
        test_num = math.floor(file_num*(1-percent))                 # number of test files/signals
        if(test_num < 1):
            test_num = 1
```

```python
        train_num = file_num-test_num                   # number of train files/signals

        for j in range(train_num):                      # traverse each train file
            train_activity = []
            cur_file = act_data[i][j]                   # current file's data
            length = cur_file.shape[0]                  # number of samples in current file
            segment_num = math.floor(length/segment_size)         # number of segment

            for k in range(segment_num):                # build up segment
                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):  # traverse each train file
            test_activity = []
            cur_file = act_data[i][j]                   # current file's data
            length = cur_file.shape[0]                  # number of samples in current file
            segment_num = math.floor(length/segment_size)         # number of segment

            for k in range(segment_num):                # build up segment
                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 createTrainHistogram(data, cluster_size, labels, segment_size):
    length = data.shape[0]
    index = 0

    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 createTestHistogram(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
```

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

    return count

def kms(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 kmsPredict(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 agg(activities, cluster_size, segment_size):
    cluster = AgglomerativeClustering(n_clusters=cluster_size, affinity='euclidean', linkage='ward')
    train_labels = cluster.fit_predict(activities[:, :segment_size*3])
    train_centers = getCenter(activities[:, :segment_size*3], train_labels, cluster_size, segment_size)
    return train_labels, train_centers

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()

def execute(act_name, act_data, segment_size, cluster_size, percent, matrix_output):
    activities, act_train, act_test = splitData(act_data, percent, segment_size)

    model, train_labels, train_centers = kms(activities, cluster_size, segment_size)
    # train_labels, train_centers = agg(activities, cluster_size, segment_size)

    train_histogram, train_signal = createTrainHistogram(act_train, cluster_size, train_labels, segment_size)
    draw(train_histogram, train_signal, cluster_size, act_name)
```

```python
    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(createTestHistogram(act_test[i], cluster_size, test_labels[i]))

    # rf = RF(max_depth=5, random_state=0).fit(train_histogram, train_signal)
    svm = SVC(gamma='auto').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 = svm.predict(test_histogram[i].reshape(1, -1))[0]
        label_ori = act_test[i][0, segment_size*3]
        # print(type(label),label,type(label_ori))
        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)
    print(segment_size, cluster_size, percent, 'kmeans', accurate/ len(act_test))
    print(cov_df, cov_df.sum())


if __name__ == "__main__":
    act_name, act_data = readData('./HMP_Dataset')

    execute(act_name, act_data, segment_size=32, cluster_size=24, percent=0.8, matrix_output=True)
```