HW5 Report

Size of the fixed length sample	Overlap (0-X%)	K- value	Classifier	Accuracy
12 * 3	0	19	RandomForest	0.77-0.8

Standard K means

Train test Split: 75%train, 25%test (The train error and test error does not have big difference, which means to avoid overfit problem)

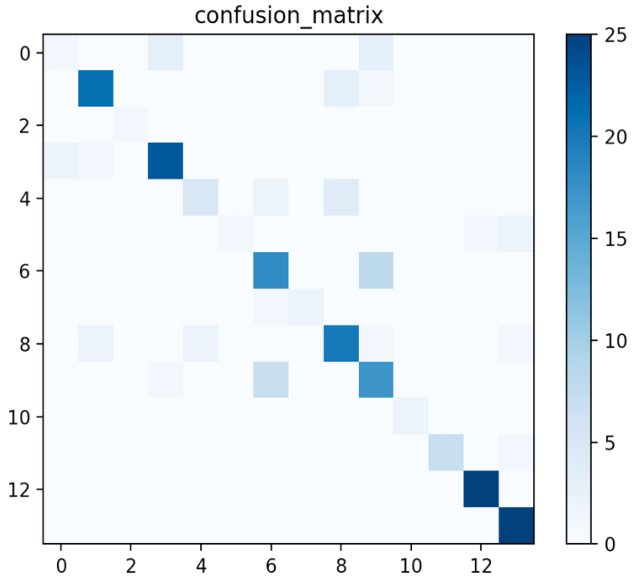
Selection of k（test when length = 32）: Selection of fixed length（test when k = 19）:

K

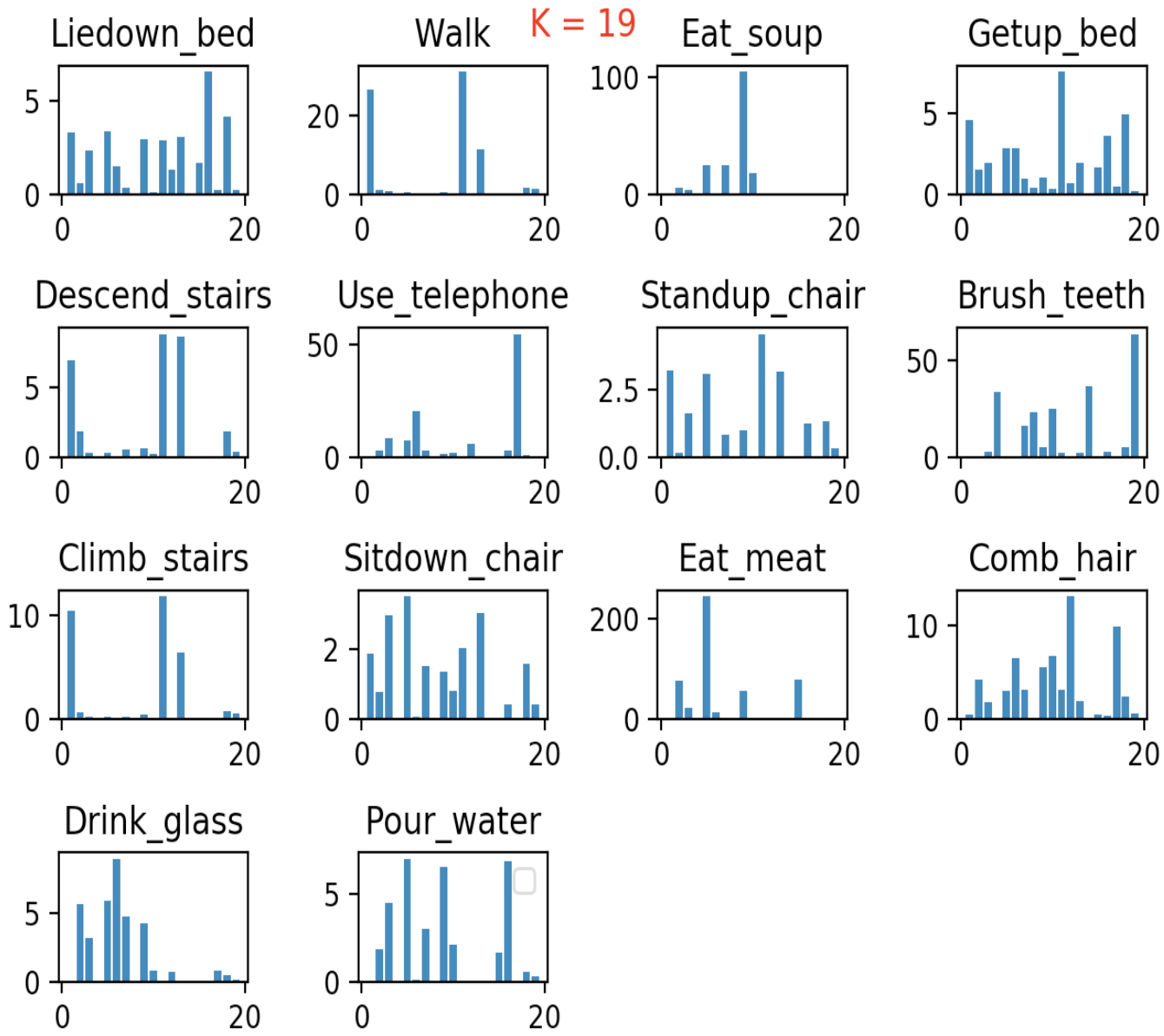
```
5 : 0.677570093457944
6 : 0.7009345794392523
7 : 0.6822429906542056
8 : 0.6635514018691588
9 : 0.6495327102803738
10 : 0.7242990654205608
11 : 0.7289719626168224
12 : 0.7523364485981309
13 : 0.6962616822429907
14 : 0.7336448598130841
15 : 0.7476635514018691
16 : 0.719626168224299
17 : 0.7663551401869159
18 : 0.719626168224299
19 : 0.7710280373831776
20 : 0.7663551401869159
21 : 0.7102803738317757
22 : 0.719626168224299
23 : 0.7336448598130841
24 : 0.705607476635514
25 : 0.7383177570093458
26 : 0.7476635514018691
27 : 0.7476635514018691
28 : 0.7149532710280374
29 : 0.7149532710280374
```

length

```
10 : 0.7616822429906542
11 : 0.7336448598130841
12 : 0.8037383177570093
13 : 0.7710280373831776
14 : 0.7850467289719626
15 : 0.7570093457943925
16 : 0.7710280373831776
17 : 0.7289719626168224
18 : 0.7476635514018691
19 : 0.705607476635514
20 : 0.7663551401869159
21 : 0.7476635514018691
22 : 0.6915887850467289
23 : 0.7242990654205608
24 : 0.7476635514018691
25 : 0.7383177570093458
26 : 0.7476635514018691
27 : 0.7523364485981309
28 : 0.7429906542056075
29 : 0.7663551401869159
```




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


K = 19
Liedown_bed
Walk
Eat_soup
Getup_bed
Descend_stairs
Use_telephone
Standup_chair
Brush_teeth
Climb_stairs
Sitdown_chair
Eat_meat
Comb_hair
Drink_glass
Pour_water

1. segmentation of the vector:

```
def divide_one_file(list, length):
    """
    Divid one file.
    """
    row_size = len(list) # The number of rows
    num_of_blocks = int(row_size / length)
    result = []
    if num_of_blocks > 0: # Make sure there is at least one block
        for idx in range(0, num_of_blocks):
            new_list = []
            start_index = idx * length
            end_index = (idx + 1) * length
            for l in range(start_index, end_index):
                new_list += list[l]
            result.append(new_list)
    return result
```

2. k-means

```
#Sum all the list
sum_all_list = []
for action in action_dict:
    for l in action_dict[action]:
        sum_all_list.append(l)
sum_all_list = np.array(sum_all_list)


#Cluster with KMeans(we use 14 clusters here)
kmeans = KMeans(n_clusters=k, random_state=0).fit(sum_all_list)
cluster_center = kmeans.cluster_centers_
```

3. generating the histogram

```
    ############ Plot Histograms ###########

    fig = plt.figure()

    action_index = 1
    for action in action_center:
        size = len(original[action])
        center_list = action_center[action]
        center_list = np.array(center_list) / size

        plt.subplot(4, 4, action_index)
        plt.bar([i for i in range(1, k + 1)], center_list)
        plt.title(action)

        action_index += 1

    fig.suptitle('K = 19', color = 'r')
    fig.tight_layout()
    plt.legend()
    plt.show()
```

4. classification

```
    clf = RandomForestClassifier(n_estimators=100, max_depth=32)
    clf.fit(train, train_label)
    #Get the result
    print(clf.score(test_data, test_label))


    ##### confusion_matrix #####
    y_predict = clf.predict(test_data)
    y_true = test_label
    cm = confusion_matrix(y_true, y_predict)
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title("confusion_matrix")
    plt.colorbar()
    plt.show()
    ############################
```

Relevant Code:
Divide the file into segments

```
def divide(action_dict, length):
    """
    Divide the whole data sets.
    """
    new_action_dict = {}
    for action in action_dict:
        new_action_dict[action] = []
        for list in action_dict[action]:
            row_size = len(list) # The number of rows
            num_of_blocks = int(row_size / length)
            if num_of_blocks > 0: # Make sure there is at least one block
                for idx in range(0, num_of_blocks):
                    new_list = []
                    start_index = idx * length
                    end_index = (idx + 1) * length
                    for l in range(start_index, end_index):
                        new_list += list[l]
                    new_action_dict[action].append(new_list)
```

Generating training matrices and training labels.

```
train = []
train_label = []
for action in original:
    all_files = original[action]
    for one_list in all_files:
        div = divide_one_file(one_list, n)
        result = [0 for i in range(k)]
        for curr_action in div:
            label = 0
            min_distance = sys.maxsize
            for center in center_mark:
                curr_center = list(center)
                distance = la.norm(np.array(curr_center) - np.array(curr_action))
                if distance < min_distance:
                    min_distance = distance
                    label = center_mark[center]
            result[label] += 1
        train.append(result)
        train_label.append(action_num[action])
```