AML HW5 Yu Che Wang/ yuchecw2

Table

In the following experiments, I use standard K-means clustering and 20%-80% test-train split using cross validation from the sklearn package. For test-train split, I preprocess the data first (cut the signals to different lengths for quantization), and then I split each class of data into testing and training data. Thus, for each class, there are 20% of them belong to testing data. However, some classes (ex: class 6 and 7) have few data resulting in a small amount of testing data for that class.

Changing K-value. The classifier has highest accuracy when K=50.

Size of the fixed length sample	Overlap (0-X%)	K-value	Classifier	Accuracy
30	0	10	Random forest n_estimators=20, max_depth=10	0.76616
		30		0.78109
		50		0.82089
		70		0.79104
		100		0.75621
		200		0.79104
		500		0.78109

Changing overlap percentage. The classifier has highest accuracy when overlap is around 30%-70%.

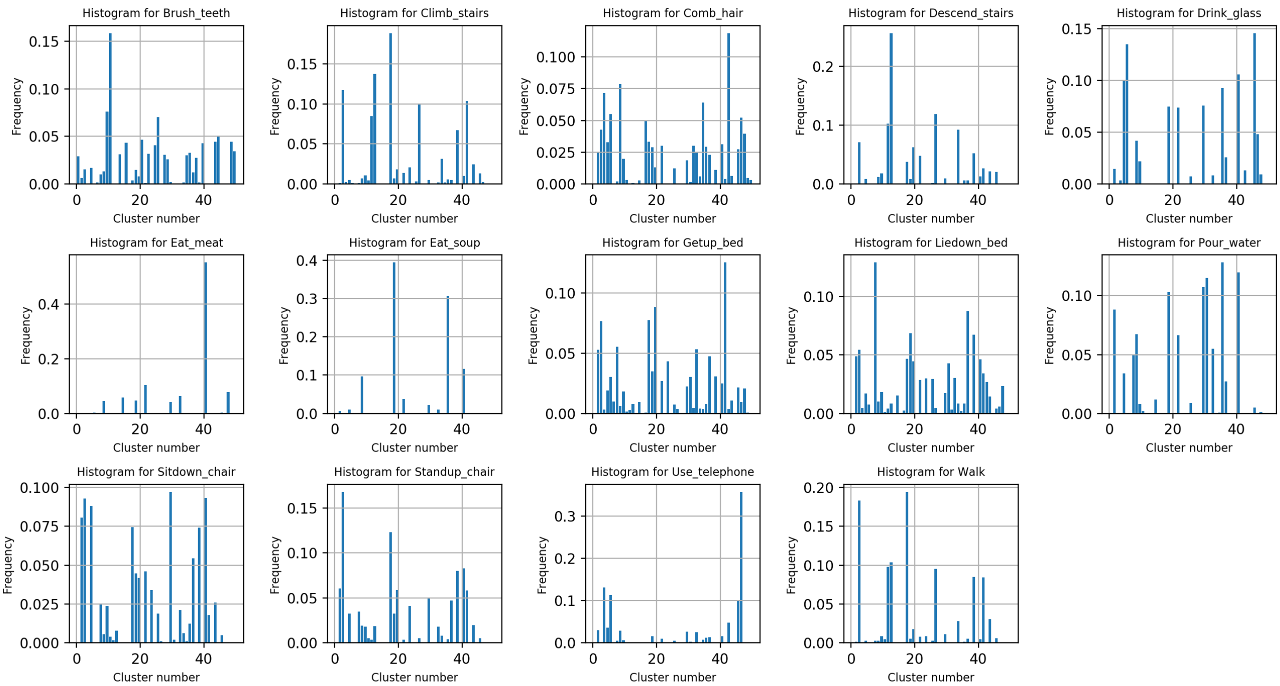
Size of the fixed length sample	Overlap (0-X%)	K-value	Classifier	Accuracy
30	0	50	Random forest n_estimators=20, max_depth=10	0.77114
	5			0.80099
	10			0.78109
	20			0.78606
	30			0.83582
	40			0.83084
	50			0.82587
	60			0.84079
	70			0.81592
	80			0.79104

Changing size of fixed length sample. The classifier has highest accuracy when the size of fixed length sample is 15. Note that for fixed length sample of size 15, we cut the accelerometer data into 5 frames, each frame with 3 channels (X,Y,Z), and then we horizontally stack the data into a 1 by 15 array.

Size of the fixed length sample	Overlap (0-X%)	K-value	Classifier	Accuracy
15	50	50	Random forest n_estimators=20, max_depth=10	0.86069
30				0.78109
45				0.78109
60				0.77611

Histograms of the mean quantized vector

Use standard K-means clustering with K=50, overlap=50%, and fixed length sample of size 15. We obtain an accuracy of 0.82. Note that we obtain different accuracies in different experiments due to random seed.



Class confusion matrix

		Actual class													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Predicted class	1	3	0	0	0	0	0	0	0	0	0	0	0	0	0
	2	0	21	0	1	0	0	0	0	0	0	0	0	0	8
	3	0	0	6	0	0	0	0	0	0	0	0	0	0	0
	4	0	1	0	8	0	0	0	0	0	0	0	0	0	0
	5	0	0	0	0	24	0	0	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	8	0	0	1	0	0	0	0	23	4	0	0	2	0	0
	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	10	0	0	0	0	0	1	1	0	0	24	0	0	1	0
	11	0	0	0	0	0	0	0	0	1	0	22	5	0	0
	12	0	0	0	0	0	0	0	2	1	0	2	18	0	2
	13	0	0	0	0	0	0	0	0	0	0	0	0	2	0
	14	0	3	0	0	0	0	0	0	0	0	0	0	0	14

Class numbers:
1: Brush_teeth; 2: Climb_stairs; 3: Comb_hair; 4: Descend_stairs; 5: Drink_glass;
6: Eat_meat; 7: Eat_soup; 8: Getup_bed; 9: Liedown_bed; 10: Pour_water;
11: Sitdown_chair; 12: Standup_chair; 13: Use_telephone; 14: Walk

Code Snippets

1. segmentation of the vector

```
def cut_signal(signal, dim, overlap):
    i=0
    signal_patches = []
    while(i+dim < signal.shape[0]):
        signal_patches.append(signal[i:i+dim, :].reshape(1, -1))
        i += (int)(dim*(1-overlap)) # no overlap
    if i < signal.shape[0]:
        signal_patches.append(signal[-dim:, :].reshape(1, -1))
    return signal_patches
```

2. k-means

```
def main(dim, n_clusters, overlap, fig):
    folders = os.listdir('HMP_Dataset')
    folders.pop(0) # Remove '.DS_Store'

    data_dict = build_data_dict(folders, dim, overlap)
    name_label_dict = build_name_label_dict(data_dict)

    train_data_dict, test_data_dict = split_data_dict(data_dict)
    train_data, train_label = get_data_label(train_data_dict, data_dict, name_label_dict) #train_data[i,:] -> train_label[i]
    test_data, test_label = get_data_label(test_data_dict, data_dict, name_label_dict)

    kmeans = KMeans(n_clusters=n_clusters, init='k-means++', n_init=5).fit(train_data)
    predict_hist, predict_label = prediction(kmeans, train_data_dict, n_clusters, name_label_dict)
```

3. generating the histogram

```
# Build frequency histogram with length n_clusters
def build_histogram(labels, n_clusters):
    histogram = [0]*n_clusters
    total = 0
    for label in labels:
        histogram[label] += 1
        total += 1
    X = np.array(histogram).reshape(1,-1)
    X = X / total
    return X
```

4. classification

```
clf = RandomForestClassifier(n_estimators=20, max_depth=10, random_state=0)
clf.fit(predict_hist, predict_label)
```

Code

Helper functions

```python
import os
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.cross_validation import train_test_split
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import argparse

def class_confusion_matrix(predict, actual):
    confusion_matrix = np.zeros((14,14)) # len(name_label_dict.keys()) = 14
    for i in range(len(predict)):
        confusion_matrix[predict[i], actual[i]] += 1
    return confusion_matrix

# Can also be used to obtain test_data, test_label
def get_data_label(train_data_dict, data_dict, name_label_dict):
    train_data =  np.vstack(train_data_dict['Brush_teeth'])
    train_label = [name_label_dict['Brush_teeth']]*train_data.shape[0]
    for key in data_dict.keys():
        if key is not 'Brush_teeth':
            current_train_data = np.vstack(train_data_dict[key])
            train_data = np.vstack([train_data, current_train_data])
            train_label += [name_label_dict[key]]*current_train_data.shape[0]
    return train_data, train_label

def build_name_label_dict(data_dict):
    name_label_dict = {}
    for i, key in enumerate(data_dict.keys()):
        name_label_dict[key] = i
    return name_label_dict

def split_data_dict(data_dict, test_size=0.20, random_state=42):
    train_data_dict = {}
    test_data_dict = {}
    for label in data_dict.keys():
        train_data, test_data = train_test_split(data_dict[label], test_size=test_size, random_state=random_state)
        train_data_dict[label] = train_data
        test_data_dict[label] = test_data
    return train_data_dict, test_data_dict
```

```python
# (label, data(shape: (n_samples, n_features))) pair
def build_data_dict(folders, dim, overlap):
    data_dict = {}
    for folder in folders:
        if folder in data_dict.keys():
            data_dict[folder] += [preprocess('HMP_Dataset/'+folder+'/'+file, dim, overlap) for file in os.listdir('HMP_Dataset/'+folder) if not file.startswith('.')]
        else:
            key_exists = False
            for key in data_dict.keys():
                if folder.startswith(key):
                    key_exists = True
                    data_dict[key] += [preprocess('HMP_Dataset/'+folder+'/'+file, dim, overlap) for file in os.listdir('HMP_Dataset/'+folder) if not file.startswith('.')]
            if not key_exists:
                data_dict[folder] = [preprocess('HMP_Dataset/'+folder+'/'+file, dim, overlap) for file in os.listdir('HMP_Dataset/'+folder) if not file.startswith('.')]
    return data_dict

# Build frequency histogram with length n_clusters
def build_histogram(labels, n_clusters):
    histogram = [0]*n_clusters
    total = 0
    for label in labels:
        histogram[label] += 1
        total += 1
    X = np.array(histogram).reshape(1,-1)
    X = X / total
    return X

# Build a histogram of shape (n_clusters,)
def prediction(kmeans, train_data_dict, n_clusters, name_label_dict):
    predict_hist = []
    predict_label = []
    for key in train_data_dict.keys():
        for data in train_data_dict[key]:
            hist = build_histogram(kmeans.predict(data), n_clusters)
            predict_hist.append(hist)
            predict_label.append(name_label_dict[key])
    predict_hist = np.array(predict_hist).squeeze(1)
    return predict_hist, predict_label
```

```python
def preprocess(file, dim, overlap):
    signal = np.loadtxt(file)
    signal_patches = cut_signal(signal, dim, overlap)
    return np.vstack(signal_patches)

def cut_signal(signal, dim, overlap):
    i=0
    signal_patches = []
    while(i+dim < signal.shape[0]):
        signal_patches.append(signal[i:i+dim, :].reshape(1, -1))
        i += (int)(dim*(1-overlap)) # no overlap
    if i < signal.shape[0]:
        signal_patches.append(signal[-dim:, :].reshape(1, -1))
    return signal_patches
```

Main function

```python
def main(dim, n_clusters, overlap, fig):
    folders = os.listdir('HMP_Dataset')
    folders.pop(0) # Remove '.DS_Store'

    data_dict = build_data_dict(folders, dim, overlap)
    name_label_dict = build_name_label_dict(data_dict)

    train_data_dict, test_data_dict = split_data_dict(data_dict)
    train_data, train_label = get_data_label(train_data_dict, data_dict, name_label_dict) #train_data[i,:] -> train_label[i]
    test_data, test_label = get_data_label(test_data_dict, data_dict, name_label_dict)

    kmeans = KMeans(n_clusters=n_clusters, init='k-means++', n_init=5).fit(train_data)
    predict_hist, predict_label = prediction(kmeans, train_data_dict, n_clusters, name_label_dict)
    clf = RandomForestClassifier(n_estimators=20, max_depth=10, random_state=0)
    clf.fit(predict_hist, predict_label)

    #Test
    test_predict_hist, test_actual_label = prediction(kmeans, test_data_dict, n_clusters, name_label_dict)
    test_predict_label = clf.predict(test_predict_hist)
    n_correct = len(test_predict_label[test_predict_label==test_actual_label])
    accuracy = n_correct / len(test_predict_label)
    print('Accuracy: {}'.format(accuracy))
    confusion_matrix = class_confusion_matrix(test_predict_label, test_actual_label)

    for key_num, key in enumerate(train_data_dict.keys()):
        sample = np.zeros((1,dim*3))
        for i in range(len(train_data_dict[key])):
            for j in range(train_data_dict[key][i].shape[0]):
                sample = np.vstack((sample, train_data_dict[key][i][j,:].reshape(1,dim*3)))

        hist = build_histogram(kmeans.predict(sample[1:,:]), n_clusters)
        ax = fig.add_subplot(3,5,key_num+1)
        ax.bar(np.arange(1, n_clusters+1)-0.4, hist.squeeze(0).tolist(), width=0.8)
        ax.set_xlabel('Cluster number', fontsize=8)
        ax.set_ylabel('Frequency', fontsize=8)
        ax.set_title('Histogram for {}'.format(key), fontsize=8)
        ax.grid()
    print(confusion_matrix)
```

Execution code

```python
fig = plt.figure(figsize=(13,7))
confusion_matrix = main(dim=5, n_clusters=50, overlap=0.5, fig=fig)
fig.tight_layout()
plt.show()
```