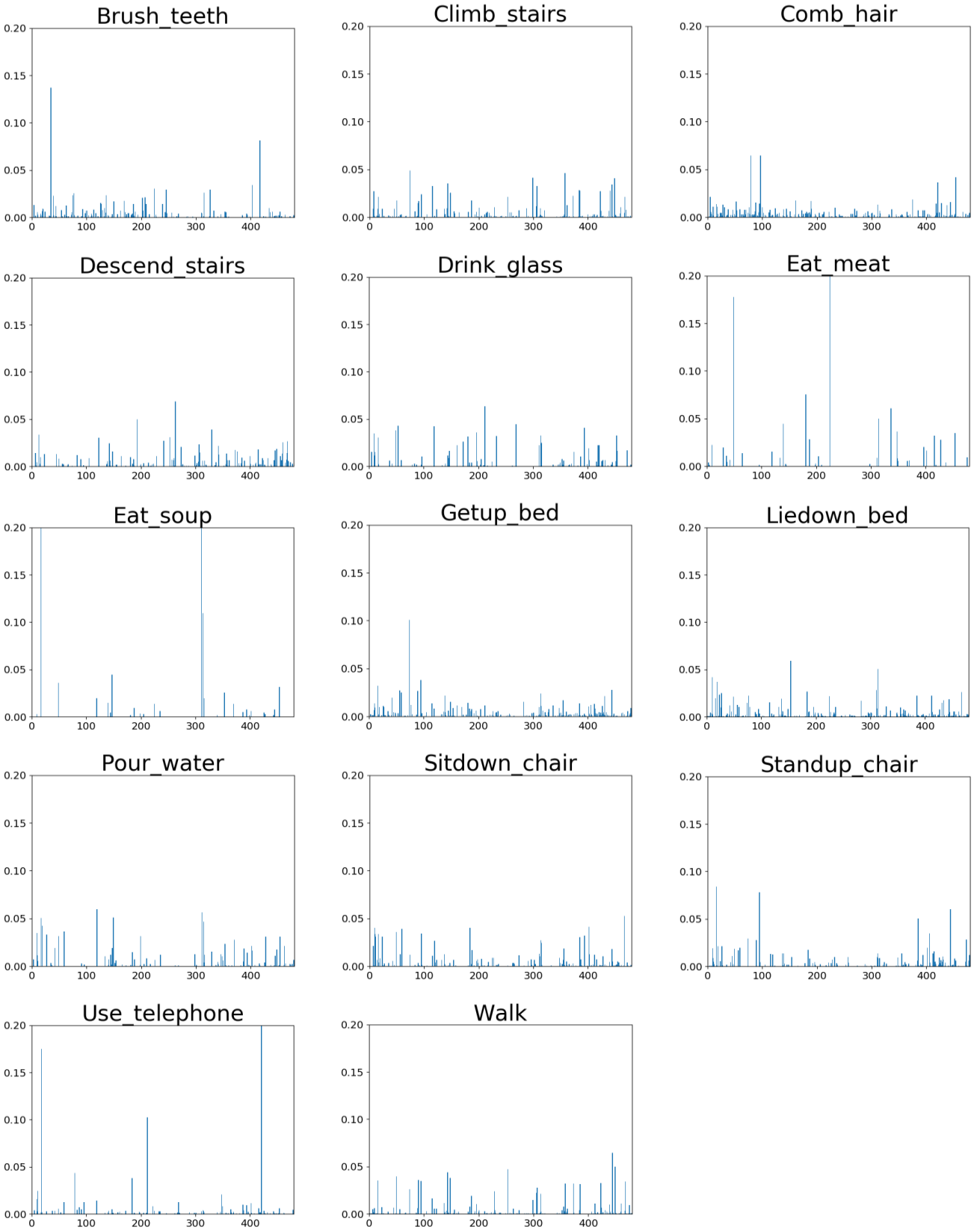
## **Page1: Experiment Table**

| **Fixed Sample Length** | **Overlap %** | **K-value (number of cluster centers)** | **Classifier** | **Classifier Accuracy** |
|---|---|---|---|---|
| 48 | 80% | 480 | random forest | 0.8091 |
| 48 | 80% | 400 | random forest | 0.7626 |
| 32 | 50% | 400 | random forest | 0.74739 |
| 32 | 80% | 480 | random forest | 0.78762 |

Note: We used Minibatch k-means for clustering

# Page 2: Histograms

K-value for best accuracy: 480



# Page 3: Confusion Matrix

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

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 2 | 0 | 0 | 1 | 0 | 0 | 4 | 0 | 0 | 0 | 3 |
| 1 | 0 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 2 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 0 | 27 | 0 | 0 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 11 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 1 |
| 6 | 0 | 0 | 0 | 1 | 0 | 0 | 31 | 0 | 0 | 2 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 2 | 0 | 2 | 0 | 0 | 2 | 0 | 27 | 1 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 33 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 2 | 0 |
| 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 34 | 0 |
| 13 | 0 | 0 | 0 | 1 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 31 |

# Page 4: A screenshot of your code

## i. Segmentation of the vector

```
def segment_data(data,size=32,overlap=0.75):
    seg_data = []
    overlap=int(size*overlap)
    for entry in data:
        seg_entry = None
        i=0
        while (i+size)<len(entry):
            seg = entry[i:i+size].T.flatten()
            if seg_entry is None:
                seg_entry=np.array(seg)
            else:
                seg_entry=np.vstack((seg_entry,seg))
            i=i+(size-overlap)
        remained = entry[-size:].T.flatten()
        seg_entry=np.vstack((seg_entry,remained))
        seg_data.append(seg_entry)
    return seg_data
```

## ii. K-means

```
cluster_model = MiniBatchKMeans(n_clusters=cluster_size, batch_size=3200, random_state=0)
```

## iii. Generating the histogram

```
def make_histogram(new_data,seg_data_label,labelDict,cluster_size):
    print("ploting histogram")
    labels = np.unique(seg_data_label)
    for i in labels:
        data_by_label=new_data[seg_data_label==i]
        mean_hist=np.mean(data_by_label,axis=0)
        plt.figure()
        plt.bar(np.arange(new_data.shape[1]),height= mean_hist,width=1.5)
        plt.ylim(0,0.2)
        plt.yticks([0,0.05,0.1,0.15,0.2],fontsize=14)
        plt.xlim(0,cluster_size)
        plt.xticks([0,100,200,300,400],fontsize=14)
        plt.title(labelDict[i],fontsize=30)
        plt.savefig(labelDict[i])
```

## iv. Classification

```
    # CLASSIFICATION
    print('Building and testing classifier ...')
    acc = [0,0,0]
    class_model = RandomForestClassifier(n_estimators=100,max_depth=10,random_state=0)

    for i in range(3):
        class_model.fit(X_train[i], y_train[i])
        y_pred = class_model.predict(X_test[i])
        print("confusion matrix:",confusion_matrix(y_test[i], y_pred))
        acc[i] = accuracy_score(y_test[i], y_pred)
    print('Accuracy: ',acc)
    print('Average Accuracy: ',np.mean(acc))
```

# Page 5: Screenshots of all Source Code

```
import os
import numpy as np
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import KMeans
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import MiniBatchKMeans
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

def load_data(path):
    folders = os.listdir(path)
    data = []
    labels = []
    labelDict={}
    for i,folder in enumerate(folders):
        folder_path = "" + path + "/" + folder
        if not os.path.isdir(folder_path):
            continue
        files = os.listdir(folder_path)
        labelDict[i]=folder
        for j,file in enumerate(files):
            labelDict[i]=folder
            file_path = "" + folder_path + "/" + file
            entry = np.loadtxt(file_path, delimiter=" ")
            data.append(entry)
            labels.append(i)
    labels = np.array(labels)
    print(labelDict)
    return data, labels, labelDict
```

```
#data: (rows_n,4),where data[i,3] is the label
def segment_data(data,size=32,overlap=0.75):
    seg_data = []
    overlap=int(size*overlap)
    for entry in data:
        seg_entry = None
        i=0
        while (i+size)<len(entry):
            seg = entry[i:i+size].T.flatten()
            if seg_entry is None:
                seg_entry=np.array(seg)
            else:
                seg_entry=np.vstack((seg_entry,seg))
            i=i+(size-overlap)
        remained = entry[-size:].T.flatten()
        seg_entry=np.vstack((seg_entry,remained))
        seg_data.append(seg_entry)
    return seg_data

def segData_to_clusterData(seg_data):
    cluster_data = None
    for seg_entry in seg_data:
        if cluster_data is None:
            cluster_data = seg_entry # ignore vector of labels
        else:
            cluster_data = np.vstack((cluster_data,seg_entry))
    return cluster_data

# seg_data: a list of segmented data
def segData_to_features(seg_data,model,cluster_size):
    feature_matrix = None
    for seg_entry in seg_data:
        feature_vec = np.zeros((1,cluster_size))
        clusters = model.predict(seg_entry)
        unique, counts = np.unique(clusters, return_counts=True)
        feature_vec[0,unique] = counts/sum(counts)
        if feature_matrix is None:
            feature_matrix = feature_vec
        else:
            feature_matrix = np.vstack((feature_matrix,feature_vec))
    return feature_matrix
```

```
def make_histogram(new_data,seg_data_label,labelDict,cluster_size):
    print("ploting histogram")
    labels = np.unique(seg_data_label)
    for i in labels:
        data_by_label=new_data[seg_data_label==i]
        mean_hist=np.mean(data_by_label,axis=0)
        plt.figure()
        plt.bar(np.arange(new_data.shape[1]),height= mean_hist,width=1.5)
        plt.ylim(0,0.2)
        plt.yticks([0,0.05,0.1,0.15,0.2],fontsize=14)
        plt.xlim(0,cluster_size)
        plt.xticks([0,100,200,300,400],fontsize=14)
        plt.title(labelDict[i],fontsize=30)
        plt.savefig(labelDict[i])


def main():

    # LOAD DATA (signals & their corresponding labels)
    print('Loading data ...')
    path = '/Users/shaohenglai/Dropbox/CS498/MP5/HMP_Dataset'
    data,y,labelDict = load_data(path)
    # VECTOR QUANTIZATION of ENTIRE DATASET
    # segment signals into vectors
    print('Segmenting data ...')
    seg_data = segment_data(data,size=48,overlap=0.8)
    # do clustering to get a vocabolary of features
    cluster_size = 480
    clustering_data = segData_to_clusterData(seg_data)
    print('Clustering data shape: ',clustering_data.shape)
    print('Training clutering model ...')
    cluster_model = MiniBatchKMeans(n_clusters=cluster_size, random_state=0)
    #cluster_model = KMeans(n_clusters = cluster_size, init = 'k-means++', random_state = 42)
    #cluster_model = AgglomerativeClustering(n_clusters = cluster_size, affinity = 'euclidean', linkage = 'ward')
    cluster_model.fit(clustering_data)
    # covert signals to feature vectors of fixed cluster_size
    print('Coverting signals to features ...')
    X = segData_to_features(seg_data,cluster_model,cluster_size)
    print('Features Matrix shape: ',X.shape)
    make_histogram(X,y,labelDict)
    # GET K-FOLDS DATA for CROSS VALIDATION
    print('Dividing data into 3-folds ...')
    X_train, X_test = [None,None,None], [None,None,None]
    y_train, y_test = [None,None,None], [None,None,None]
    skf = StratifiedKFold(n_splits=3)
    for i,(train_index, test_index) in enumerate(skf.split(X, y)):
            X_train[i], X_test[i] = X[train_index], X[test_index]
            y_train[i], y_test[i] = y[train_index], y[test_index]

    # CLASSIFICATION
    print('Building and testing classifier ...')
    acc = [0,0,0]
    class_model = RandomForestClassifier(n_estimators=100,max_depth=10,random_state=0)

    for i in range(3):
        class_model.fit(X_train[i], y_train[i])
        y_pred = class_model.predict(X_test[i])
        print("confusion matrix:",confusion_matrix(y_test[i], y_pred))
        acc[i] = accuracy_score(y_test[i], y_pred)
    print('Accuracy: ',acc)
    print('Average Accuracy: ',np.mean(acc))

main()
```