**CS498 AMO Homework 4**

Team :

Minyuan Gu ([minyuan3@illinois.edu](mailto:minyuan3@illinois.edu), netid minyuan3)

Yanislav Shterev ([shterev2@illinois.edu](mailto:shterev2@illinois.edu), netid shterev2)

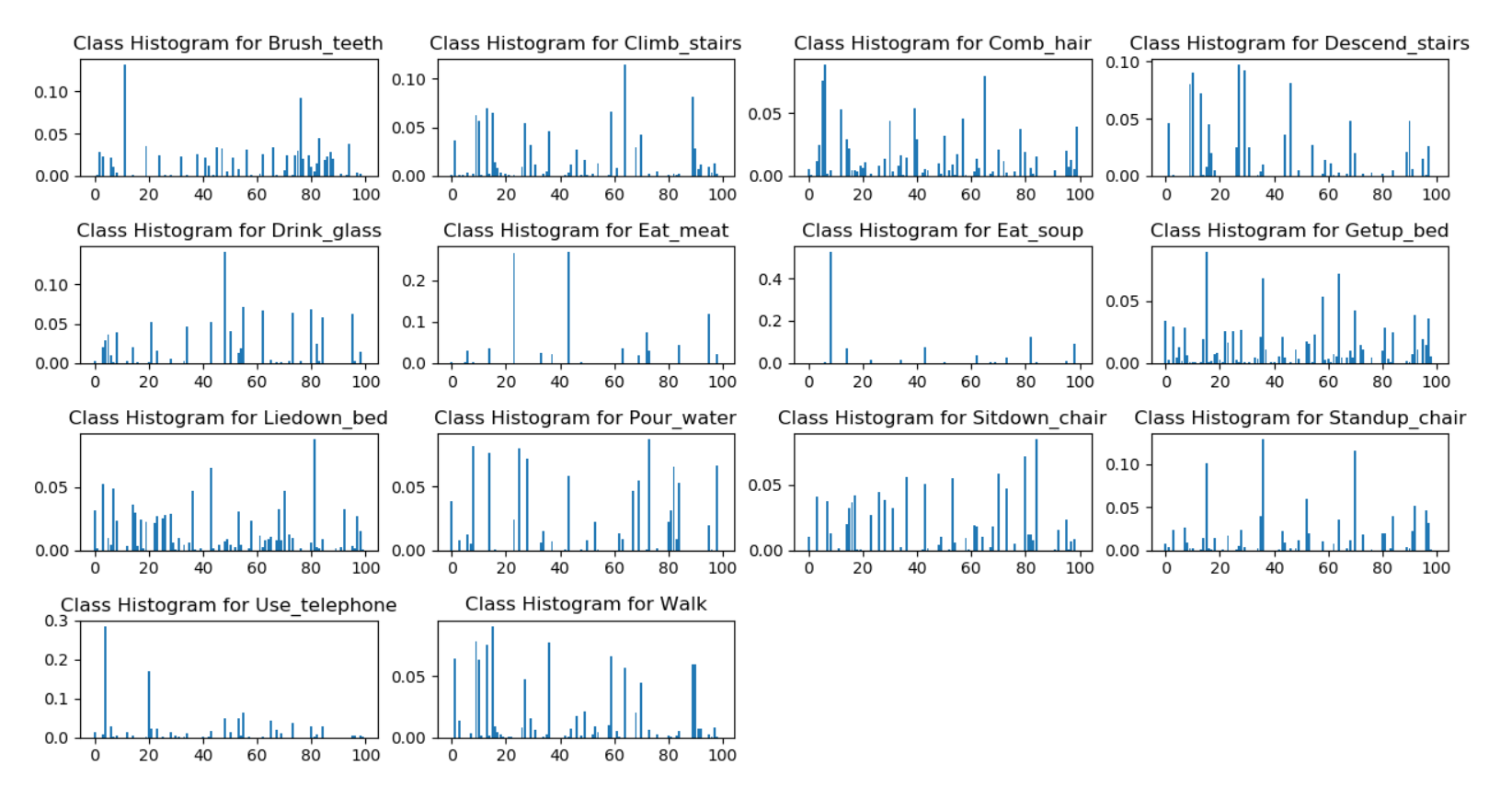
**1. Page 1 (40 pts)** **Experiment table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fixed length size** | **Overlap(0-X%)** | **K-value** | **Classifier** | **Accuracy** |
| 32 | 70% | 100 | Standard | 0.8105 |
| 32 | 70% | 110 | Standard | 0.8140 |
| 32 | 70% | 120 | Standard | 0.7906 |
| 48 | 70% | 100 | Standard | 0.8233 |
| 48 | 70% | 110 | Standard | 0.8491 |
| 48 | 70% | 120 | Standard | 0.8210 |
| 32 | 20% | 100 | Standard | 0.7918 |
| 32 | 20% | 110 | Standard | 0.8035 |
| 32 | 20% | 120 | Standard | 0.7964 |
| 48 | 20% | 100 | Standard | 0.7684 |
| 48 | 20% | 110 | Standard | 0.7801 |
| 48 | 20% | 120 | Standard | 0.7543 |
| 32 | 70% | 100 | Hierarchical | 0.7777 |
| 32 | 70% | 110 | Hierarchical | 0.7637 |
| 32 | 70% | 120 | Hierarchical | 0.7766 |
| 48 | 70% | 100 | Hierarchical | 0.7789 |
| 48 | 70% | 110 | Hierarchical | 0.7368 |
| 48 | 70% | 120 | Hierarchical | 0.7345 |

**2. Page 2 (28 pts) Histograms**

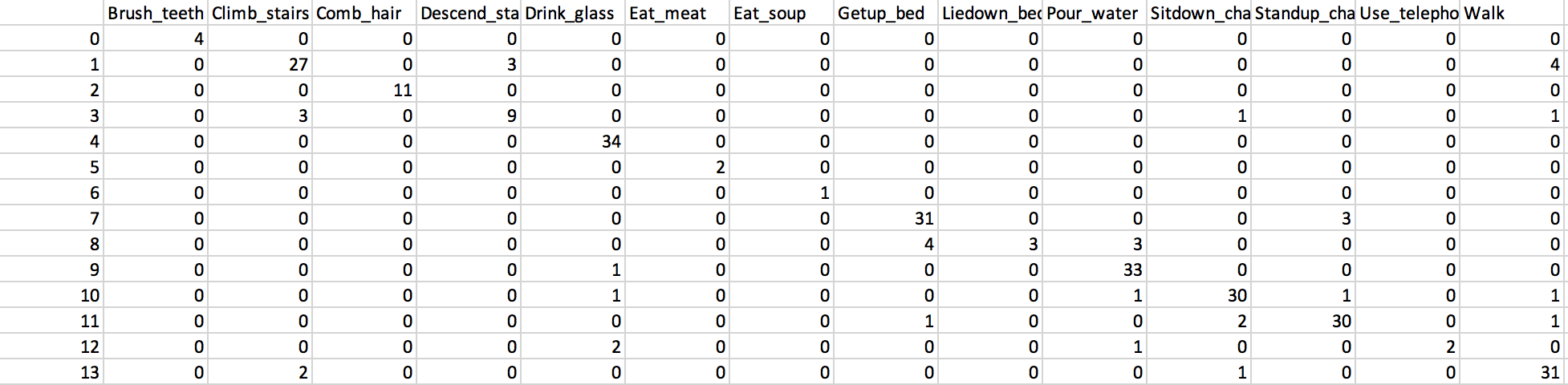
Histograms of the mean quantized vector (Histogram of cluster centres like in the book) for each activity with the K value that gives you the highest accuracy.

Segment size= 48, K= 100, 70% overlap and standard K-means. The accuracy achieved based on this configuration is 0.8491.



**3. Page 3 (22 pts)** **Confusion matrix**

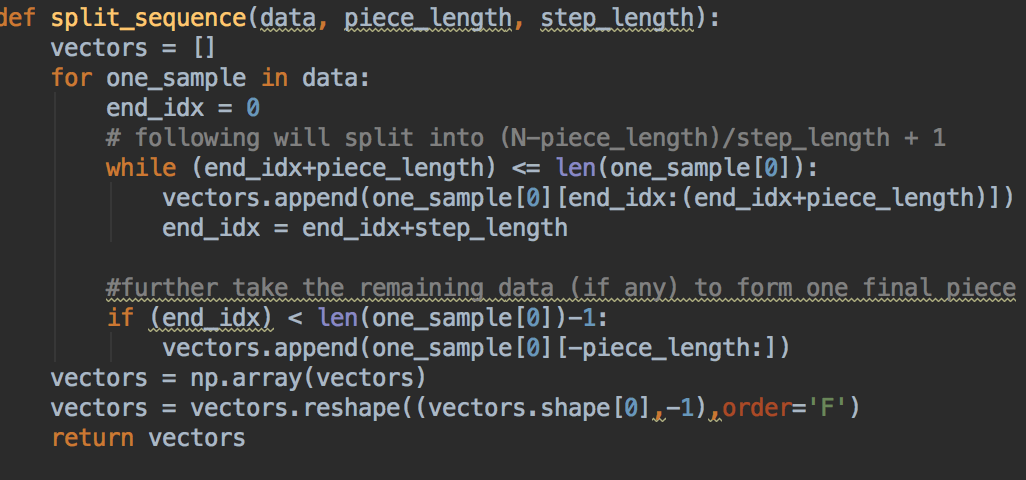
Class confusion matrix from the classifier that you used. Please make sure to label the row/colums of the matrix so that we know which row corresponds to what.



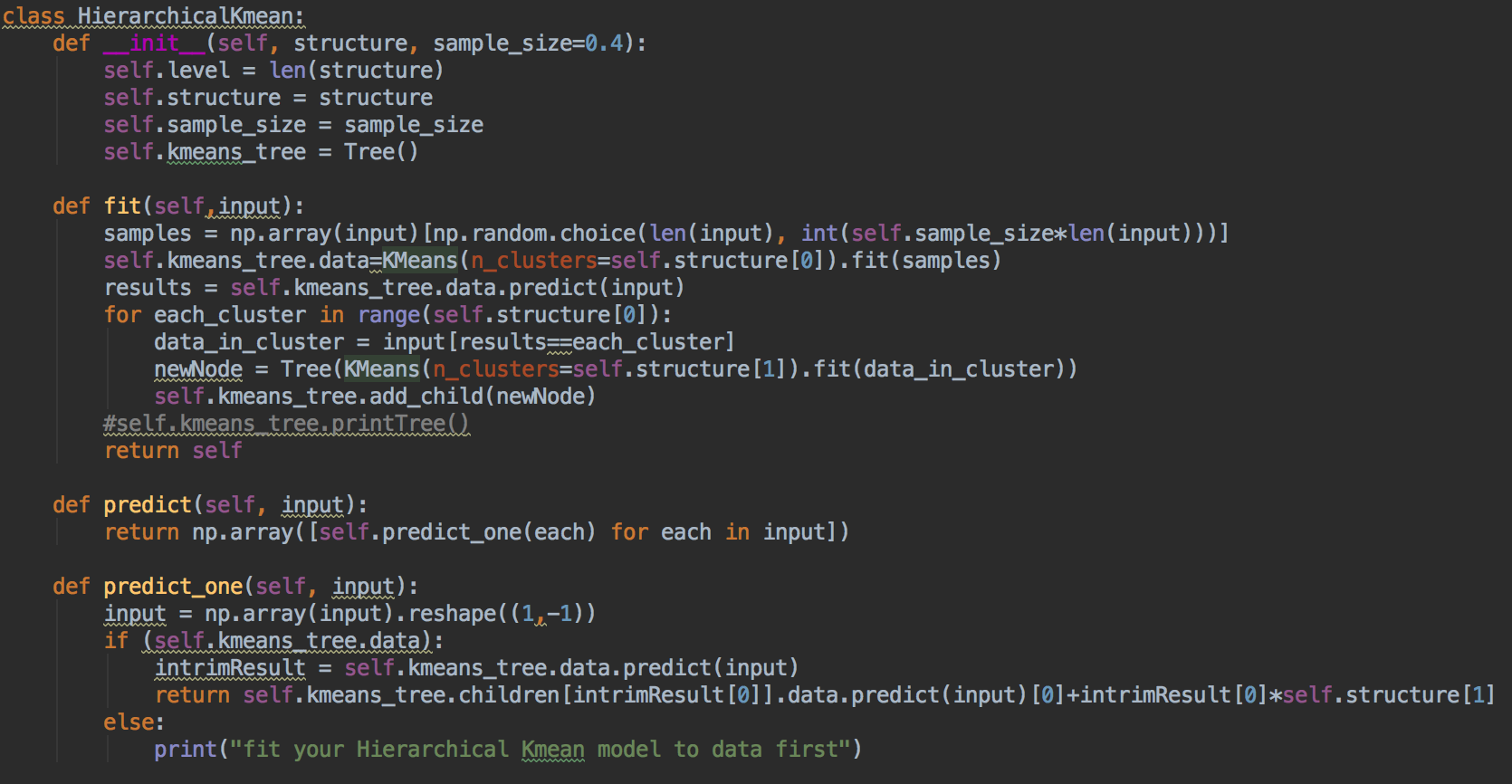
**4. Page 4 (10 pts)** **A screenshot of your code**

The page should contain snippets of code demonstrating:

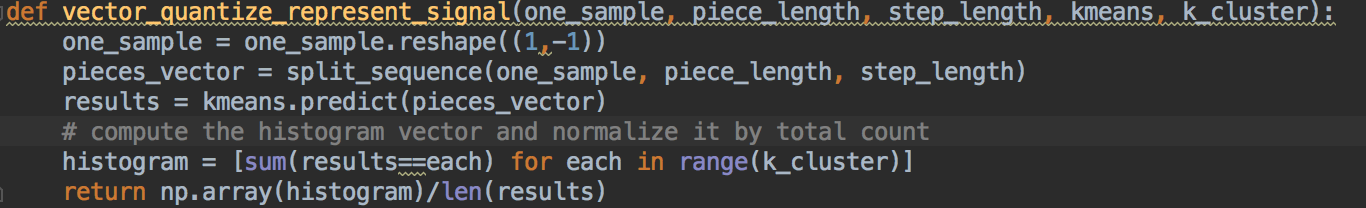
1. Segmentation of the vector



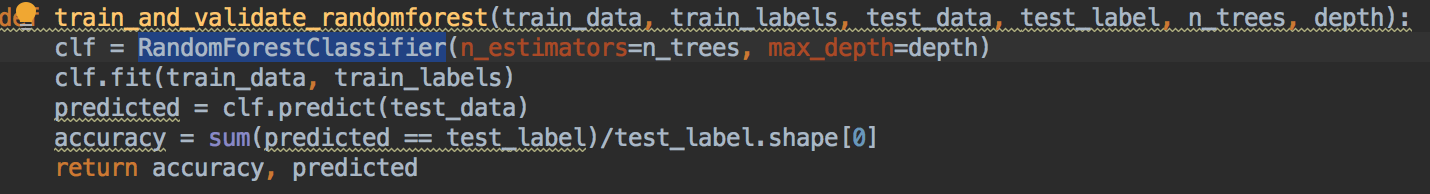
1. K-means



1. Generating the histogram



    iv) Classification



**5. Page 5+ Screenshots of all your source code.**

import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.cluster import KMeans  
from sklearn.ensemble import RandomForestClassifier  
import matplotlib.pyplot as plt  
from sklearn.metrics import confusion\_matrix  
import os  
import pandas as pd  
  
activity = ["Brush\_teeth", "Climb\_stairs", "Comb\_hair", "Descend\_stairs",  
 "Drink\_glass", "Eat\_meat", "Eat\_soup", "Getup\_bed",  
 "Liedown\_bed", "Pour\_water", "Sitdown\_chair", "Standup\_chair",  
 "Use\_telephone", "Walk"]  
  
class Tree:  
 def \_\_init\_\_(self, data=None):  
 self.children = []  
 self.data = data  
 def add\_child(self, node):  
 self.children.append(node)  
 def printTree(self):  
 print('node data is', self.data)  
 for each in self.children:  
 each.printTree()  
  
  
  
class HierarchicalKmean:  
 def \_\_init\_\_(self, structure, sample\_size=0.4):  
 self.level = len(structure)  
 self.structure = structure  
 self.sample\_size = sample\_size  
 self.kmeans\_tree = Tree()  
  
 def fit(self,input):  
 samples = np.array(input)[np.random.choice(len(input), int(self.sample\_size\*len(input)))]  
 self.kmeans\_tree.data=KMeans(n\_clusters=self.structure[0]).fit(samples)  
 results = self.kmeans\_tree.data.predict(input)  
 for each\_cluster in range(self.structure[0]):  
 data\_in\_cluster = input[results==each\_cluster]  
 newNode = Tree(KMeans(n\_clusters=self.structure[1]).fit(data\_in\_cluster))  
 self.kmeans\_tree.add\_child(newNode)  
 #self.kmeans\_tree.printTree()  
 return self  
  
 def predict(self, input):  
 return np.array([self.predict\_one(each) for each in input])  
  
 def predict\_one(self, input):  
 input = np.array(input).reshape((1,-1))  
 if (self.kmeans\_tree.data):  
 intrimResult = self.kmeans\_tree.data.predict(input)  
 return self.kmeans\_tree.children[intrimResult[0]].data.predict(input)[0]+intrimResult[0]\*self.structure[1]  
 else:  
 print("fit your Hierarchical Kmean model to data first")  
  
  
def split\_sequence(data, piece\_length, step\_length):  
 vectors = []  
 for one\_sample in data:  
 end\_idx = 0  
 # following will split into (N-piece\_length)/step\_length + 1  
 while (end\_idx+piece\_length) <= len(one\_sample[0]):  
 vectors.append(one\_sample[0][end\_idx:(end\_idx+piece\_length)])  
 end\_idx = end\_idx+step\_length  
  
 #further take the remaining data (if any) to form one final piece  
 if (end\_idx) < len(one\_sample[0])-1:  
 vectors.append(one\_sample[0][-piece\_length:])  
 vectors = np.array(vectors)  
 vectors = vectors.reshape((vectors.shape[0],-1),order='F')  
 return vectors  
  
  
def compute\_clusters(vectors, k\_cluster, hierarchical=False, hierarch\_structure = None):  
 if hierarchical and hierarch\_structure is not None:  
 kmeans = HierarchicalKmean(hierarch\_structure).fit(vectors)  
 else:  
 kmeans = KMeans(n\_clusters=k\_cluster).fit(vectors)  
 return kmeans  
  
def vector\_quantize\_build\_dictionary(data, piece\_length, step\_length, k\_cluster, hierarchical=False, hierarch\_structure = None):  
 pieces\_vectors=split\_sequence(data, piece\_length, step\_length)  
 kmeans = compute\_clusters(pieces\_vectors, k\_cluster, hierarchical, hierarch\_structure)  
 return kmeans  
  
def vector\_quantize\_represent\_signal(one\_sample, piece\_length, step\_length, kmeans, k\_cluster):  
 one\_sample = one\_sample.reshape((1,-1))  
 pieces\_vector = split\_sequence(one\_sample, piece\_length, step\_length)  
 results = kmeans.predict(pieces\_vector)  
 # compute the histogram vector and normalize it by total count  
 histogram = [sum(results==each) for each in range(k\_cluster)]  
 return np.array(histogram)/len(results)  
  
def quantize\_all\_data(data, piece\_length, step\_length, kmeans, k\_cluster):  
 vectorized\_data = [np.concatenate([vector\_quantize\_represent\_signal(one\_sample, piece\_length, step\_length, kmeans, k\_cluster),[one\_sample[1]]]) for one\_sample in data]  
 return np.array(vectorized\_data)  
  
  
def plot\_histogram(plt, vectorized\_data, activity\_id):  
 bins = vectorized\_data.shape[1]-1 # reduce by 1 which is the label.  
 vectors = vectorized\_data[vectorized\_data[:,-1] == activity\_id][:,:-1]  
 mean\_vector = np.mean(vectors, axis=0)  
 plt.bar(np.arange(bins), mean\_vector)  
 plt.title("Class Histogram for "+activity[activity\_id])  
 plt.show()  
  
def train\_and\_validate\_randomforest(train\_data, train\_labels, test\_data, test\_label, n\_trees, depth):  
 clf = RandomForestClassifier(n\_estimators=n\_trees, max\_depth=depth)  
 clf.fit(train\_data, train\_labels)  
 predicted = clf.predict(test\_data)  
 accuracy = sum(predicted == test\_label)/test\_label.shape[0]  
 return accuracy, predicted  
  
  
def split\_training\_test\_set(input\_data, folds):  
 test\_set = []  
 train\_set = []  
 for idx, each\_class in enumerate(activity):  
 current\_activity = input\_data[input\_data[:,-1]==idx]  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(current\_activity[:,:-1], current\_activity[:,-1], test\_size=1/folds)  
 train\_set.append(np.column\_stack((X\_train, y\_train)))  
 test\_set.append(np.column\_stack((X\_test, y\_test)))  
 train\_set = np.concatenate(train\_set)  
 np.random.shuffle(train\_set)  
 test\_set = np.concatenate(test\_set)  
 return train\_set, test\_set  
  
  
data = []  
for i, \_ in enumerate(activity):  
 files = os.listdir("HMP\_Dataset/"+activity[i])  
 for file in files:  
 sequence\_data = []  
 fobj = open("HMP\_Dataset/"+activity[i]+"/"+file, "r")  
 for line in fobj:  
 fields = line.split()  
 sequence\_data.append(fields)  
 data.append(np.array([np.array(sequence\_data).astype(float),i]))  
data = np.array(data)  
  
accuracy\_list = []  
best\_accuracy = 0  
kmean\_list = None  
piece\_length\_list = [ 48]  
k\_cluster\_list = [100]  
#hierarch\_struct = [[40,12],[40, 15], [40,20]]  
#hierarch\_struct = [[10,3],[10, 4], [10,5], [10, 6], [10, 7]]  
#hierarch\_struct = [[10,4],[16, 10], [40,6], [40, 12]]  
#hierarch\_struct = [[40,6]]  
  
#piece\_length = 32 # 32hz per second, so we take just 1 second of data into a piece.  
  
skip\_portion = 0.3 # overlap% will be 1-skip\_portion  
  
for piece\_length in piece\_length\_list:  
 step\_length = int(piece\_length\*skip\_portion)  
 for idx, k\_cluster in enumerate(k\_cluster\_list):  
 kmeans = vector\_quantize\_build\_dictionary(data,piece\_length, step\_length, k\_cluster)  
 #kmeans = vector\_quantize\_build\_dictionary(data, piece\_length, step\_length, k\_cluster, hierarchical=True, hierarch\_structure=hierarch\_struct[idx])  
 vectorized\_data = quantize\_all\_data(data, piece\_length, step\_length, kmeans, k\_cluster)  
 repeats = 3  
 average\_accuracy = 0  
 for iterate in range(repeats):  
 train\_st, test\_st = split\_training\_test\_set(vectorized\_data,3)  
 accuracy, predicted = train\_and\_validate\_randomforest(train\_st[:,:-1], train\_st[:,-1], test\_st[:,:-1], test\_st[:,-1], 120, 30)  
 #print('iteration #',iterate," accuracy is: ", accuracy)  
 average\_accuracy += accuracy  
 cm = []  
 if accuracy >= best\_accuracy:  
 kmean\_list=kmeans  
 cm = confusion\_matrix(test\_st[:, -1], predicted)  
 print(confusion\_matrix(test\_st[:, -1], predicted))  
 print('Best accuracy is: ', accuracy)  
 best\_accuracy = accuracy  
  
 average\_accuracy /= repeats  
 print('piece\_length:', piece\_length, ' K:', k\_cluster, ' Average Accuracy is: ', average\_accuracy)  
 accuracy\_list.append([piece\_length, k\_cluster, average\_accuracy])  
  
  
  
#fobj = open("./homework5/accuracy\_trees80\_depth32.csv",'a+')  
fobj = open("accuracy.csv",'a+')  
for row in np.array(accuracy\_list):  
 #fobj.write(', '.join(row.astype(str)) + ',Hierarchical-Kmean,'+str((1-skip\_portion)\*100)+'%\n')  
 fobj.write(', '.join(row.astype(str)) + ',standard,'+str((1-skip\_portion)\*100)+'%\n')  
fobj.close()  
  
  
def plot\_all\_histogram(vectorized\_data):  
 for idx, \_ in enumerate(activity):  
 plt.subplot(4, 4, idx+1)  
 plot\_histogram(plt, vectorized\_data, idx)  
 plt.subplots\_adjust(hspace=0.6)  
 plt.show()  
  
def plot\_confusion\_matrix(matrix):  
 fig, ax = plt.subplots()  
 for i in range(len(matrix)):  
 for j in range(len(matrix)):  
 c = matrix[i][j]  
 ax.text(i + 0.5, j + 0.5, str(c), va='center', ha='center')  
 ax.set\_xlim(0, len(matrix))  
 ax.set\_ylim(0, len(matrix))  
 ax.set\_xticks(np.arange(len(matrix)))  
 ax.set\_yticks(np.arange(len(matrix)))  
 ax.grid()  
  
plot\_all\_histogram(vectorized\_data)

**Libraries used & Reference:**

**David Forsyth’s book** - Applied Machine Learning

**Trevor Walker’s lecture and sample code** – CS-498 Lecture videos

**Accelerometer dataset** - <https://archive.ics.uci.edu/ml/datasets/Dataset+for+ADL+Recognition+with+Wrist-worn+Accelerometer>

**Numpy** - <http://www.numpy.org/>

**Sklearn**

* train test split - <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html>
* KMeans – to generate the cluster centers: <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>
* RandomForestClassifier – to train classifier and predict <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
* Confusion\_matrix: <https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html>

**matplotlib.pyplot** - <https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html>