**CS498 AMO Homework 5**

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**

We tested combinations of different parameters and we listed a few typical settings below for discussion:

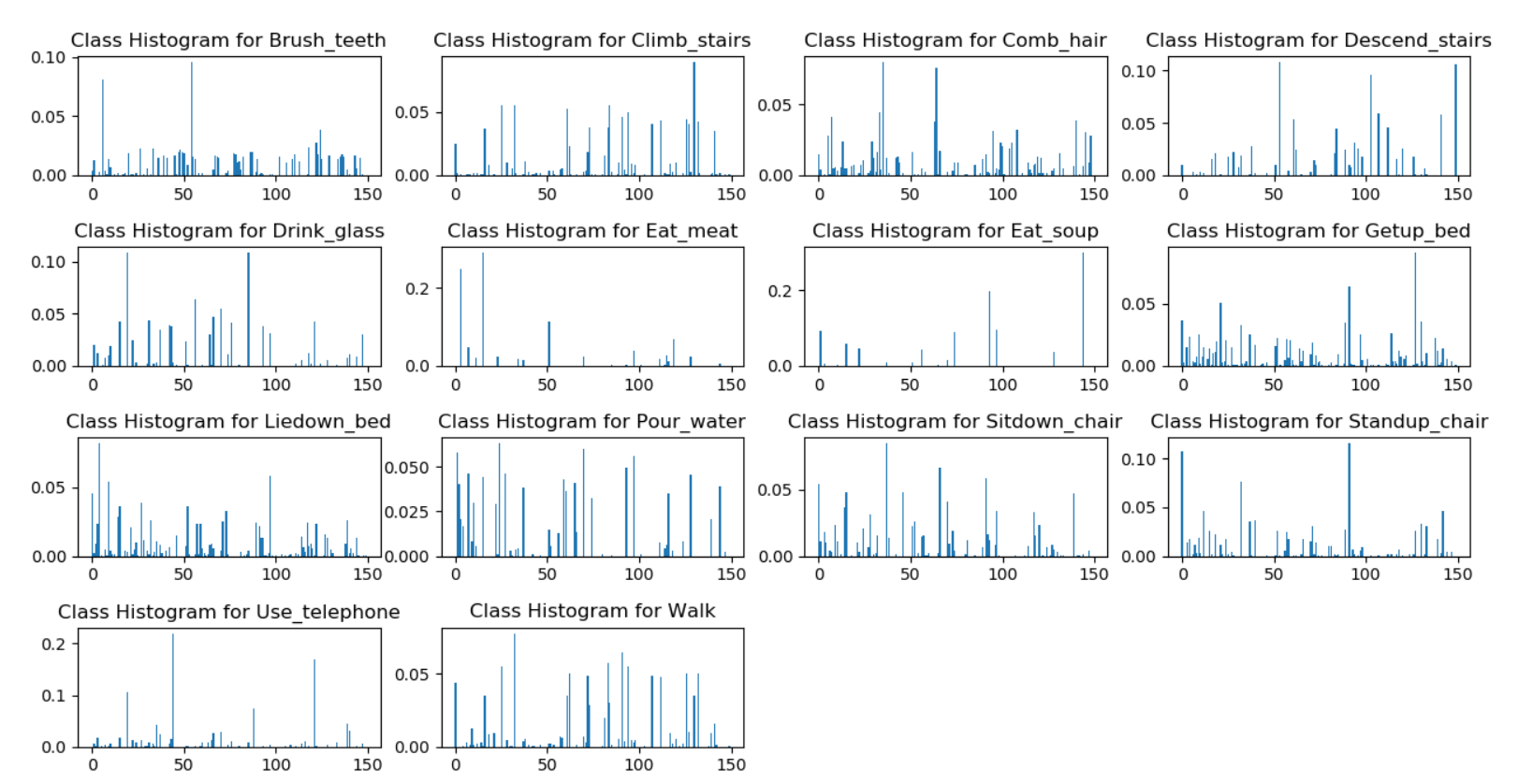
1. We tried different segment sizes (16 – 48) and they all have similar accuracy and size 32 and 48 were chosen since they provided good results and serves as a sliding window of 1~1.5 seconds of the information from activities.
2. Overlap X% has big influence on the accuracy. Our code introduced parameters (skip\_portion & step\_length = piece\_length\*skip\_portion) to control it. Higher overlapping percentage achieved better accuracy but longer execution time. This is easily understood: smaller the move of the sliding window (segment) better the chance to capture more decisive patterns; small steps creates large amounts of segments which make the execution time much longer. We tested 0% (no overlap), 30%, 50%, 70% & 90% overlap.
3. We tried different sets of K values, ranging from 40 to 800. We chose values between 100~300 which yielded good accuracy > 83%. We tested but did not opt for large K value, since it takes longer time to train without obvious accuracy gain (e.g. >300)
4. Hierarchical K-mean makes large K efficient - it speeded up our algorithm execution time but we didn’t observe significant improvement when choosing a large number of K.
5. Apart from above improvement-seeking we did, we also tested different classifiers (K-neighbor & Random forest), also different settings of random forest. You can refer to the below table.



**2. Page 2 (28 pts) Histograms**

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

We used: Segment size= 48, K= 150, 90% overlap and standard K-means. The best accuracy achieved based on this configuration is 87.72% on a single validation.



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

Class confusion matrix from the classifier that we used.

From below it is obvious the classifier works quite well on most of the activities, except for ‘Lie down bed’ which was classified mostly as ‘sit down chair’ which make sense given both are similar activities from Accelerometer’s perspective.



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

Note: We added some side notes to explain the code, hoping to assist our TA/grader to save some time.

1. Segmentation of the vector

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

1. K-means

We have both Hierarchical Kmean and normal Kmean (from sklearn) to choose.

We have a common entry point

compute\_cluster()

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)  
 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 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

1. Generating the histogram

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)

Two steps to generate histogram:

1. Build dictionary (split sequence + Kmean)

2. Use dictionary to quantize vector by building histograms.

kmeans = vector\_quantize\_build\_dictionary(data,piece\_length, step\_length, k\_cluster)  
vectorized\_data = quantize\_all\_data(data, piece\_length, step\_length, kmeans, k\_cluster)

1. Classification  
   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

**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  
from sklearn.neighbors import KNeighborsClassifier  
import matplotlib.pyplot as plt  
from sklearn.metrics import confusion\_matrix  
import os  
  
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)  
 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 train\_and\_validate\_KNeighborsClassifier(train\_data, train\_labels, test\_data, test\_label, n\_neighbors=3, leafs=30):  
 clf = KNeighborsClassifier(n\_neighbors=n\_neighbors, leaf\_size=leafs)  
 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):  
 set1 = []  
 set2 = []  
 set3 = []  
 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)  
 X\_train2, X\_test2, y\_train2, y\_test2 = train\_test\_split(X\_train, y\_train, test\_size= 1 / (folds-1))  
 set1.append(np.column\_stack((X\_train2, y\_train2)))  
 set2.append(np.column\_stack((X\_test2, y\_test2)))  
 set3.append(np.column\_stack((X\_test, y\_test)))  
 set1 = np.concatenate(set1)  
 np.random.shuffle(set1)  
 set2 = np.concatenate(set2)  
 np.random.shuffle(set2)  
 set3 = np.concatenate(set3)  
 np.random.shuffle(set2)  
 return [set1, set2, set3]  
  
  
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()  
  
  
######################################################  
# Load data #  
######################################################  
data = []  
for i, \_ in enumerate(activity):  
 files = os.listdir(**"./homework5/HMP\_Dataset/"**+activity[i])  
 for file in files:  
 sequence\_data = []  
 fobj = open(**"./homework5/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)  
  
  
######################################################  
# For standard Kmean - hyper parameters tuning #  
######################################################  
piece\_length\_list = [4, 10, 16, 32, 48, 64] # 32hz per second, 32 means we take 1 second of data into a piece. 48 is for 1.5 seconds.  
k\_cluster\_list = [100, 120, 150, 300]  
  
skip\_portion = 0.1 # overlap% will be 1-skip\_portion. e.g. 50% overlap = 0.5, 70% = 0.3, 90% = 0.1  
accuracy\_list = []  
best\_accuracy = 0  
kmean\_best\_accuracy = None  
vectorized\_data = None  
# loop through different combination of the K values and piece length values.  
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)  
 vectorized\_data = quantize\_all\_data(data, piece\_length, step\_length, kmeans, k\_cluster)  
 repeats = 3  
 average\_accuracy = 0  
 sets = split\_training\_test\_set(vectorized\_data, repeats)  
 for iterate in range(repeats):  
 test\_st = sets[iterate]  
 train\_st = np.concatenate([sets[(iterate+1)%repeats], sets[(iterate+2)%repeats]])  
 accuracy, predicted = train\_and\_validate\_randomforest(train\_st[:,:-1], train\_st[:,-1], test\_st[:,:-1], test\_st[:,-1], 120, 30)  
 # uncomment below line for KNeighbors classifier testing.  
 # accuracy, predicted = train\_and\_validate\_KNeighborsClassifier(train\_st[:,:-1], train\_st[:,-1], test\_st[:,:-1], test\_st[:,-1], n\_neighbors=1, leafs=30)  
 # print('iteration #',iterate," accuracy is: ", accuracy) <--- uncomment this for debugging  
 average\_accuracy += accuracy  
 if accuracy >= best\_accuracy:  
 kmean\_best\_accuracy=kmeans  
 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])  
  
plot\_all\_histogram(vectorized\_data)  
print(kmean\_best\_accuracy)  
  
##########################################################  
# For Hierarchical Kmean - hyper parameters tuning #  
##########################################################  
piece\_length\_list = [16, 32, 48]  
k\_cluster\_list = [480, 500, 600, 800]  
hierarch\_struct = [[40, 12], [50, 10], [40, 15], [40, 20]]  
  
skip\_portion = 0.1 # overlap% will be 1-skip\_portion. e.g. 50% overlap = 0.5, 70% = 0.3, 90% = 0.1  
accuracy\_list = []  
best\_accuracy = 0  
vectorized\_data = None  
kmean\_best\_accuracy = None  
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, 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  
 sets = split\_training\_test\_set(vectorized\_data, repeats)  
 for iterate in range(repeats):  
 test\_st = sets[iterate]  
 train\_st = np.concatenate([sets[(iterate+1)%repeats], sets[(iterate+2)%repeats]])  
 accuracy, predicted = train\_and\_validate\_randomforest(train\_st[:,:-1], train\_st[:,-1], test\_st[:,:-1], test\_st[:,-1], 120, 30)  
 # uncomment below line for KNeighbors classifier testing.  
 # accuracy, predicted = train\_and\_validate\_KNeighborsClassifier(train\_st[:,:-1], train\_st[:,-1], test\_st[:,:-1], test\_st[:,-1], n\_neighbors=1, leafs=30)  
 # print('iteration #',iterate," accuracy is: ", accuracy)  
 average\_accuracy += accuracy  
 if accuracy >= best\_accuracy:  
 kmean\_best\_accuracy=kmeans  
 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])  
  
plot\_all\_histogram(vectorized\_data)  
print(kmean\_best\_accuracy)  
fobj = open(**"./homework5/accuracy.csv"**,**'a+'**)  
for row in np.array(accuracy\_list):  
 # choose one of the below lines to dump the search into a csv file.  
 #fobj.write(', '.join(row.astype(str)) + ',Hierarchical-Kmean,'+str((1-skip\_portion)\*100)+'%,RandomForest (120Trees 30 Depth)\n')  
 fobj.write(**', '**.join(row.astype(str)) + **',standard,'** + str((1 - skip\_portion) \* 100) + **'%,RandomForest (120Trees 30 Depth)**\n**'**)  
 #fobj.write(', '.join(row.astype(str)) + ',standard,' + str((1 - skip\_portion) \* 100) + '%,KNeighbors (1 neigbor 30 leafs)\n')  
fobj.close()

**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>
* KNeighbors: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier.predict>

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