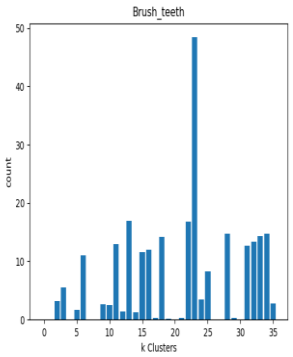
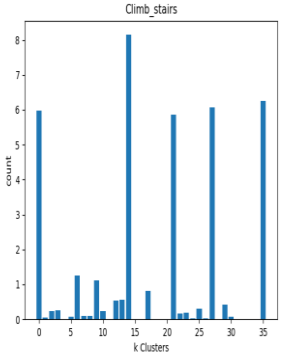
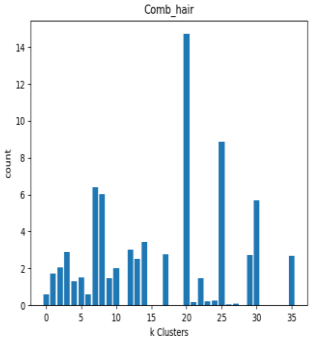
Experiment Table:

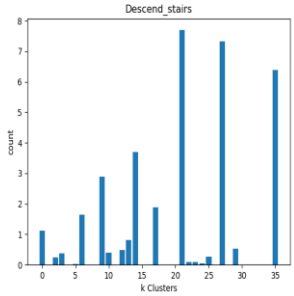
Fixed Sample Length	OverLap %	K-Value(Standard K means)	Classifier Accuracy
32	0	48	0.7318527
32	0	148	0.7318100
32	0	20	0.7318186
32	0	480	0.7008619
16	0	48	0.7366103
48	0	48	0.7198925
64	0	48	0.6912954
25	0	36	0.7485364
10	0	36	0.7735236
5	0	36	0.7723332
10	0	200	0.7501575

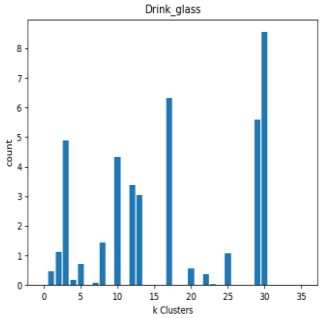
All histograms are produced using K-value of 36

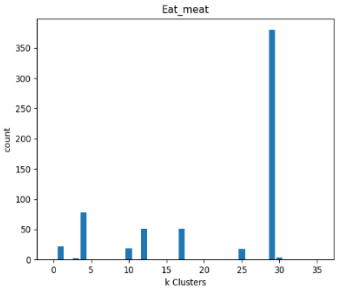


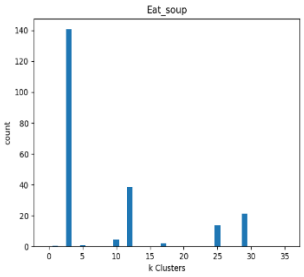


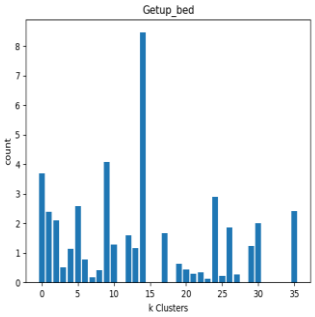


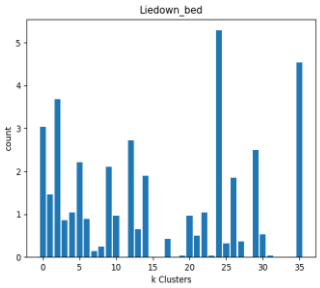


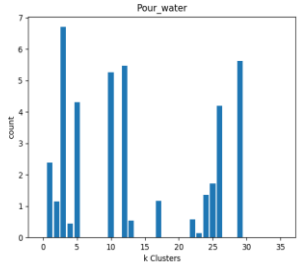


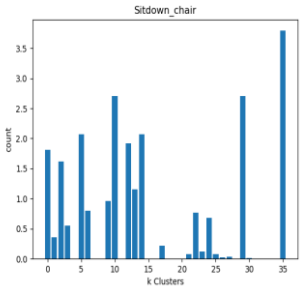


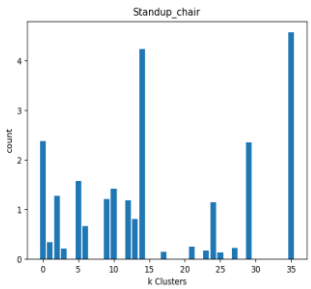


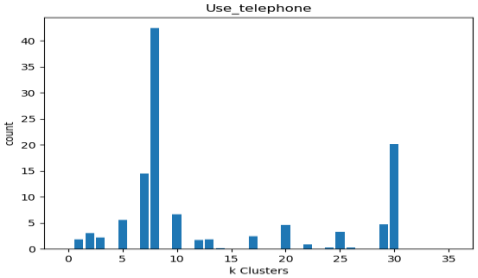


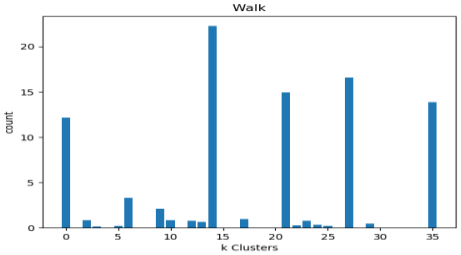












Confusion matrix:

True Label	Predicted															Error
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	
	1	3	0	0	0	0	0	0	0	0	0	0	1	0	0	25%
	2	0	27	0	4	1	0	0	1	0	0	0	0	0	1	20.6%
	3	0	0	10	0	0	0	0	0	0	0	1	0	0	0	9.1%
	4	0	4	0	8	0	0	0	0	0	0	0	1	0	1	42.9%
	5	0	0	0	0	32	0	0	0	0	1	0	0	0	0	3.0%
	6	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0.0%
	7	0	0	0	0	0	0	0	0	0	1	0	0	0	0	100%
	8	0	0	0	0	0	0	0	26	3	0	1	3	0	0	21.2%
	9	0	0	0	0	0	0	0	5	0	1	2	2	0	0	100%
	10	0	0	0	0	0	0	0	0	0	33	0	0	0	0	0.0%
	11	0	0	0	0	0	0	0	3	0	0	17	13	0	0	48.5%
	12	0	1	0	0	0	0	0	1	0	0	6	26	0	0	23.5%
	13	0	0	1	0	3	0	0	0	0	1	0	0	0	0	100%
	14	0	5	0	1	0	0	0	1	0	0	0	1	0	25	24.2%

```
Key:
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
```

i) Segmentation of Vector:

```
def getSegment(file, segmentLength):
 segments, segment = [], []
 with open(file) as f:
     size = segmentLength;
     for line in f.readlines();:
         if(size == 0):
             segments.append(segment)
             segment = []
             size = segmentLength
         segment.extend([int(x) for x in line.split()])
         size -= 1
```

ii) K-means

```
def getClusterModel(dataFiles):
  segmentsFromAllFiles = pData.getSegmentsFromDisk(allSegmentsFile,
  dataFiles, segmentLength)
  kmeans = KMeans(n_clusters = clusterCount).fit(segmentsFromAllFiles)
```

iii) Generating the histogram

```
def plotHistForAllClasses(dataFiles, kmeans):
 x = np.arange(clusterCount)
 for i in range(len(dataFiles)):
     fvs = getFeatureVectors(kmeans, dataFiles[i]);
     meanFV = np.mean(np.array(fvs), 0)
     plt.bar(x, meanFV)
     plt.title(paths[i])
     plt.ylabel("count")
     plt.xlabel("k Clusters")
     plt.show()
```

iv) Classification

```
def getLabelsUsingRandomForest(testSet, trainSet, kmeans):
 testFV, trainFV, testLabels, trainLabels, = [],[],[],[]
 for i in range(len(testSet)):
     testFV.extend(getFeatureVectors(kmeans, testSet[i]))
     testLabels.extend([i for j in range(len(testSet[i]))])
     trainFV.extend(getFeatureVectors(kmeans, trainSet[i]))
     trainLabels.extend([i for j in range(len(trainSet[i]))])

 clf = RandomForestClassifier(n_estimators = treeNum, max_depth =
maxDepths)
 clf.fit(trainFV, trainLabels)
  return clf.predict(testFV), testLabels
```

From processData.py:

```
import os
import numpy as np

#use paths provided to find all file names for each class
def getDataFiles(paths):
    classCount = len(paths)
    dataFiles = []
    for i in range(classCount):
        dataFiles.append(set())

    for i in range(classCount):
        for root, dirs, files in os.walk(paths[i]):
            for file in files:
                if "txt" in file:
                    dataFiles[i].add(root+'/'+file)
    return dataFiles

#specify a fix length and file path for all classes, open each file and cut it
into piece of segment length
def getAllSegments(segmentLength, dataFiles):
    segments = []
    for i in range(len(dataFiles)):
        for file in dataFiles[i]:
            segments.extend(getSegment(file, segmentLength))
    return segments

#given a file name, return list of fixed length segment
def getSegment(file, segmentLength):
    segments, segment = [], []
    with open(file) as f:
        size = segmentLength
        for line in f.readlines():
            if(size == 0):
                segments.append(segment)
                segment = []
                size = segmentLength
            segment.extend([int(x) for x in line.split()])
            size -= 1
    return segments

#if given source file exist, load and return it. Else compute from class path
provided, save to disk and return it
def getSegmentsFromDisk(sourceFile, dataFiles, segmentLength):
```

```
    if os.path.isfile(sourceFile):
        return np.load(sourceFile)
    else:
        segmentsFromAllFiles = np.array(getAllSegments(segmentLength,
dataFiles))
        np.save(sourceFile, segmentsFromAllFiles)
        return segmentsFromAllFiles

#given a paths to all classes, divide files in each class into n category
def splitDataFile(dataFiles, n):
    splitedDataFiles = [ [ [] for j in range(n) ] for i in
range(len(dataFiles))]
    temp = 0
    for i in range(len(dataFiles)):
        for file in dataFiles[i]:
            if(temp == n):
                temp = 0
            splitedDataFiles[i][temp].append(file)
            temp += 1
    return splitedDataFiles

#given a splited data files, return a training set and testing set, where
testing set is at index i as specified and training set is everything else
def getTrainAndTestSet(splitedDataFiles, testSetIndex):
    testSet, trainSet = [], []
    for i in range(len(splitedDataFiles)):
        testSet.append([])
        trainSet.append([])
        for j in range(len(splitedDataFiles[i])):
            if j == testSetIndex:
                testSet[i].extend(splitedDataFiles[i][j])
            else:
                trainSet[i].extend(splitedDataFiles[i][j])
    return testSet, trainSet
```

From ForestClassiferWithVQ.py:

```
import processData as pData
import numpy as np
import os
from sklearn.cluster import KMeans
from joblib import dump, load
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt

#relative path to each class folder
paths =
['Brush_teeth','Climb_stairs','Comb_hair','Descend_stairs','Drink_glass','Eat_m
eat','Eat_soup','Getup_bed','Liedown_bed','Pour_water','Sitdown_chair','Standup
_chair','Use_telephone','Walk']
#define fix size
segmentLength, clusterCount, splitCount, treeNum, maxDepths = 10, 36, 3, 30, 16
allSegmentsFile, clusterModelLib, rfModelLib = 'allSegments.npy',
'cluster.joblib', 'rfModel.joblib'
rfModelSaved = False

#load and return cluster model if already computed before, else compute, save
and return the model
def getClusterModel(dataFiles):
    if os.path.isfile(clusterModelLib):
        return load(clusterModelLib)
    else:
        segmentsFromAllFiles = pData.getSegmentsFromDisk(allSegmentsFile,
dataFiles, segmentLength)
        kmeans = KMeans(n_clusters = clusterCount).fit(segmentsFromAllFiles)
        dump(kmeans, clusterModelLib)
        return kmeans

#given a sample files, return histogram vectors from cluster model
def getFeatureVectors(kmeans, files):
    fVectors = []
    for file in files:
        fVector = [0 for i in range(kmeans.n_clusters)]
        segments = pData.getSegment(file, segmentLength)
        for label in kmeans.predict(segments):
            fVector[label] += 1
        fVectors.append(fVector)
    return fVectors
```

```
#given training and testing set, return predicted and true labels
def getLabelsUsingRandomForest(testSet, trainSet, kmeans):
    testFV, trainFV, testLabels, trainLabels, = [],[],[],[]

    for i in range(len(testSet)):
        testFV.extend(getFeatureVectors(kmeans, testSet[i]))
        testLabels.extend([i for j in range(len(testSet[i]))])
        trainFV.extend(getFeatureVectors(kmeans, trainSet[i]))
        trainLabels.extend([i for j in range(len(trainSet[i]))])

    clf = RandomForestClassifier(n_estimators = treeNum, max_depth = maxDepths)
    clf.fit(trainFV, trainLabels)

    dump(clf, rfModelLib)

    return clf.predict(testFV), testLabels

#given randomForest model and testing set, compute confusion table
def getConfusionMatrix(testSet):
    rfModel = load(rfModelLib)
    for i in range(len(testSet)):
        predictedLables = rfModel.predict(getFeatureVectors(kmeans,
testSet[i]))
        predictedCount = [0 for j in range(len(testSet))]
        for lables in predictedLables:
            predictedCount[lables] += 1
        print(predictedCount)
        print(f'Error rate is: {1-predictedCount[i]/sum(predictedCount)}')
        print()

#given predicted and true labels, return accuracy
def getAccuracy(pLabels, tLables):
    correct = 0
    for i in range(len(pLabels)):
        if pLabels[i] == tLables[i]:
            correct += 1
    return correct / len(pLabels)

#plot a average histogram for all class
def plotHistForAllClasses(dataFiles, kmeans):
    x = np.arange(clusterCount)
    for i in range(len(dataFiles)):
        fvs = getFeatureVectors(kmeans, dataFiles[i]);
        meanFV = np.mean(np.array(fvs), 0)
```

```
        plt.bar(x, meanFV)
        plt.title(paths[i])
        plt.ylabel("count")
        plt.xlabel("k Clusters")
        plt.show()

#load cluster model
dataFiles = pData.getDataFiles(paths)
kmeans = getClusterModel(dataFiles)

#plot histogram for each classes
#plotHistForAllClasses(dataFiles, kmeans)

#train and test
splitedDataFiles = pData.splitDataFile(dataFiles, splitCount)
accuracies, avgAcc = [], 0
for i in range(splitCount):
    testSet, trainSet = pData.getTrainAndTestSet(splitedDataFiles, i)
    predictedLables, trueLabels = getLabelsUsingRandomForest(testSet, trainSet,
kmeans)
    accuracies.append(getAccuracy(predictedLables, trueLabels))
avgAcc = sum(accuracies) / len(accuracies)

print(avgAcc)
```