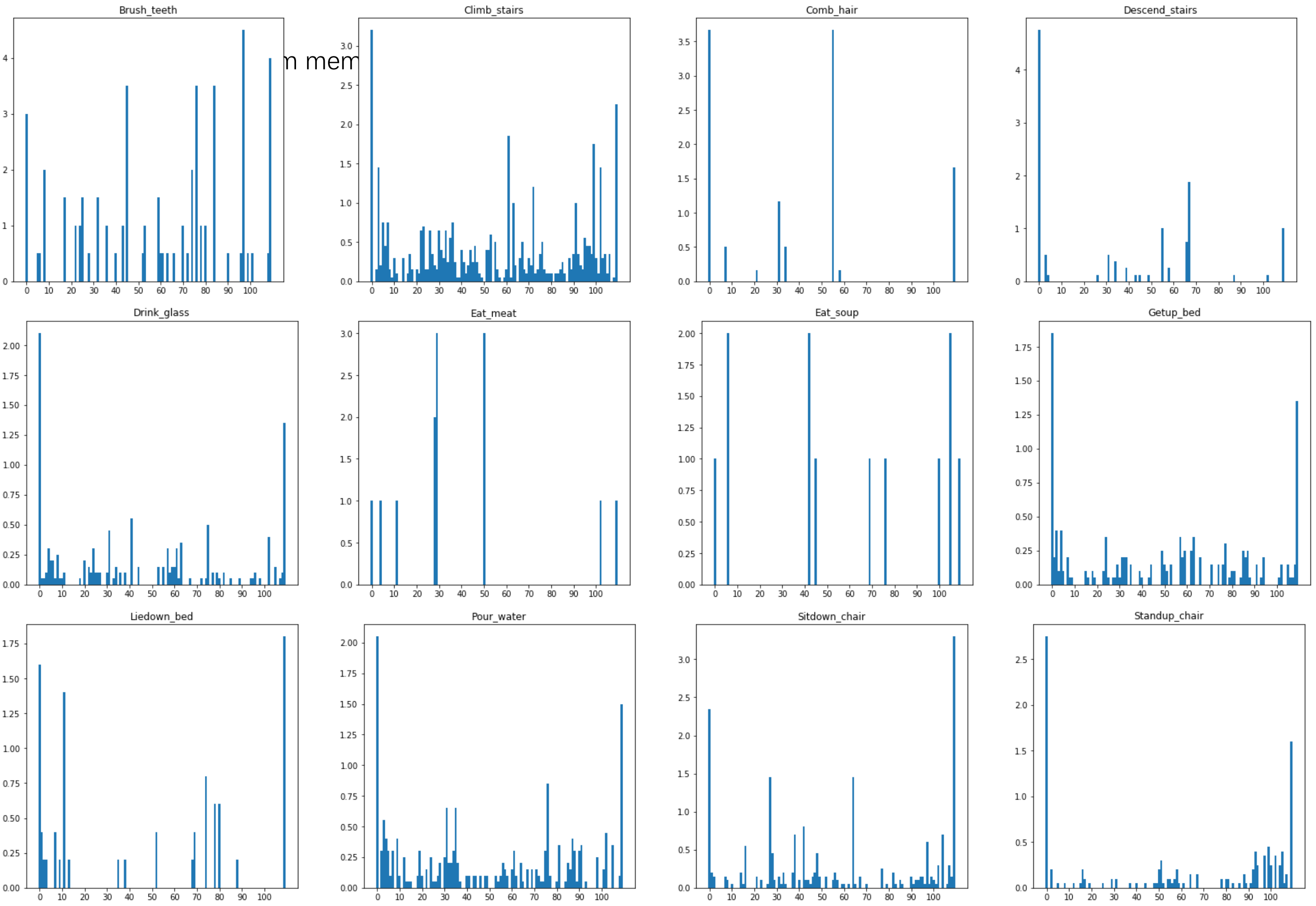
HW 5 : Team members (xt5, ts8, tanvi3)

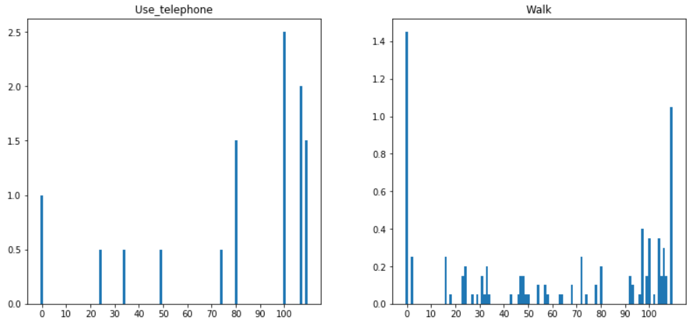
Table listing the experiments carried out:

Size of the fixed length sample	Overlap	K- value	Classifier	Accuracy
32	0	110	Support Vector Machine	0.793939
32	0	116	Support Vector Machine	0.781818
16	0	108	Random Forest	0.769697
16	0	120	Support Vector Machine	0.769697
16	0	120	Random Forest	0.769697
32	0	116	Random Forest	0.769697
16	0	108	Support Vector Machine	0.769697
32	0	106	Support Vector Machine	0.763636
32	0	102	Support Vector Machine	0.763636
16	0	100	Support Vector Machine	0.763636
32	0	110	Random Forest	0.763636
16	0	110	Support Vector Machine	0.763636
32	0	106	Multinomial Naive Bayes	0.757576
16	0	116	Support Vector Machine	0.757576
16	0	100	Random Forest	0.757576
16	0	114	Support Vector Machine	0.751515
16	0	118	Random Forest	0.751515
32	0	114	Random Forest	0.751515
32	0	116	Multinomial Naive Bayes	0.751515
32	0	102	Random Forest	0.745455
64	0	108	Multinomial Naive Bayes	0.745455
16	0	112	Support Vector Machine	0.745455
96	0	116	Multinomial Naive Bayes	0.745455
16	0	102	Support Vector Machine	0.745455
16	0	102	Random Forest	0.739394
32	0	100	Support Vector Machine	0.739394
16	0	110	Random Forest	0.739394
16	0	104	Random Forest	0.739394
16	0	116	Random Forest	0.733333
16	0	114	Random Forest	0.733333
16	0	118	Support Vector Machine	0.727273
32	0	118	Support Vector Machine	0.727273
16	0	112	Random Forest	0.721212
64	0	112	Multinomial Naive Bayes	0.721212
16	0	106	Support Vector Machine	0.721212
32	0	112	Random Forest	0.715152
32	0	100	Random Forest	0.715152
32	0	100	Multinomial Naive Bayes	0.709091
32	0	108	Support Vector Machine	0.709091
32	0	104	Random Forest	0.709091

We used **standard K-means**.
For classification, we did the training-test split on the files by **training:test = 8:2**. In other words, for each class, we select 80% files as the training data, and others as the test data. If there is less than 4 files, we randomly select a files as the test data.

n mem



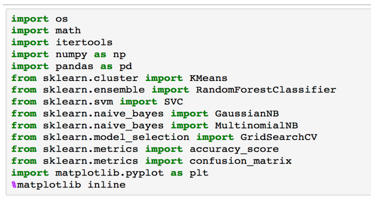


K value : 110 (for highest accuaracy)

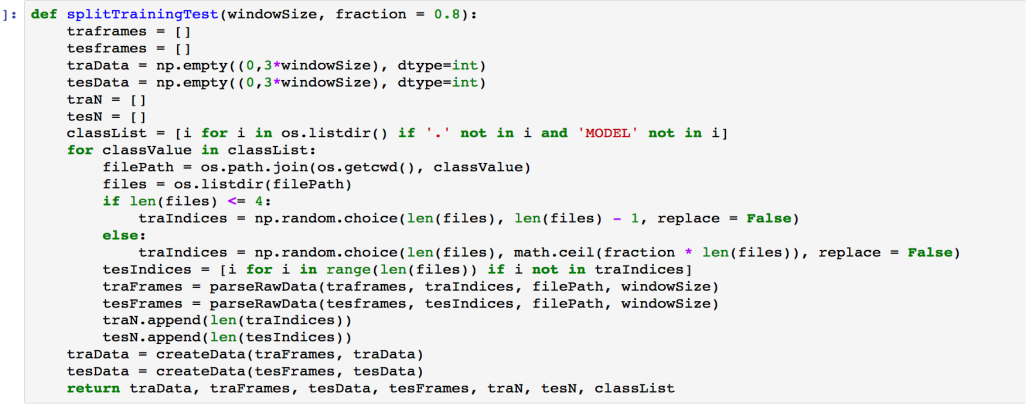
	Brush_teeth	Climb_stairs	Comb_hair	Descend_stairs	Drink_glass	Eat_meat	Eat_soup	Getup_bed	Liedown_bed	Pour_water	Sitdown_chair	Standup_chair	Use_telephone	Walk
Brush_teeth	1	0	0	0	0	0	0	0	0	0	0	0	0	1
Climb_stairs	0	16	0	0	0	0	0	0	0	0	2	1	0	1
Comb_hair	0	0	5	0	1	0	0	0	0	0	0	0	0	0
Descend_stairs	0	7	0	1	0	0	0	0	0	0	0	0	0	0
Drink_glass	0	0	0	0	20	0	0	0	0	0	0	0	0	0
Eat_meat	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Eat_soup	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Getup_bed	0	0	0	0	2	0	0	13	0	0	0	5	0	0
Liedown_bed	0	0	0	0	0	0	0	2	0	1	2	0	0	0
Pour_water	0	0	0	0	0	0	0	0	0	20	0	0	0	0
Sitdown_chair	0	0	0	0	0	0	0	2	0	0	11	6	0	1
Standup_chair	0	0	0	0	0	0	0	4	0	0	7	9	0	0
Use_telephone	0	0	0	0	0	0	0	0	0	0	0	0	1	1
Walk	0	1	0	0	0	0	0	0	0	0	0	0	0	19

HW 5 : Team members (xt5, ts8, tanvi3)

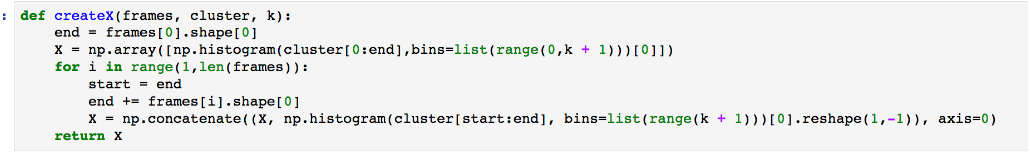
Code Snippets:



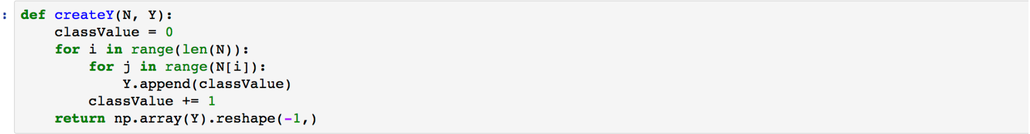
```
import os
import math
import itertools
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
%matplotlib inline
```



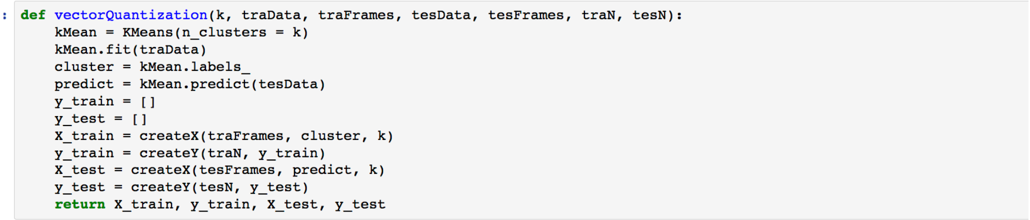
```
def splitTrainingTest(windowSize, fraction = 0.8):
    traframes = []
    tesframes = []
    traData = np.empty((0,3*windowSize), dtype=int)
    tesData = np.empty((0,3*windowSize), dtype=int)
    traN = []
    tesN = []
    classList = [i for i in os.listdir() if '.' not in i and 'MODEL' not in i]
    for classValue in classList:
        filePath = os.path.join(os.getcwd(), classValue)
        files = os.listdir(filePath)
        if len(files) <= 4:
            traIndices = np.random.choice(len(files), len(files) - 1, replace = False)
        else:
            traIndices = np.random.choice(len(files), math.ceil(fraction * len(files)), replace = False)
        tesIndices = [i for i in range(len(files)) if i not in traIndices]
        traFrames = parseRawData(traframes, traIndices, filePath, windowSize)
        tesFrames = parseRawData(tesframes, tesIndices, filePath, windowSize)
        traN.append(len(traIndices))
        tesN.append(len(tesIndices))
    traData = createData(traFrames, traData)
    tesData = createData(tesFrames, tesData)
    return traData, traFrames, tesData, tesFrames, traN, tesN, classList
```



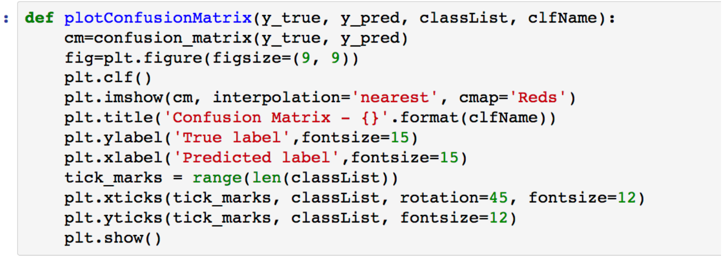
```
def createX(frames, cluster, k):
    end = frames[0].shape[0]
    X = np.array([np.histogram(cluster[0:end],bins=list(range(0,k + 1)))[0]])
    for i in range(1,len(frames)):
        start = end
        end += frames[i].shape[0]
        X = np.concatenate((X, np.histogram(cluster[start:end], bins=list(range(k + 1)))[0].reshape(1,-1)), axis=0)
    return X
```



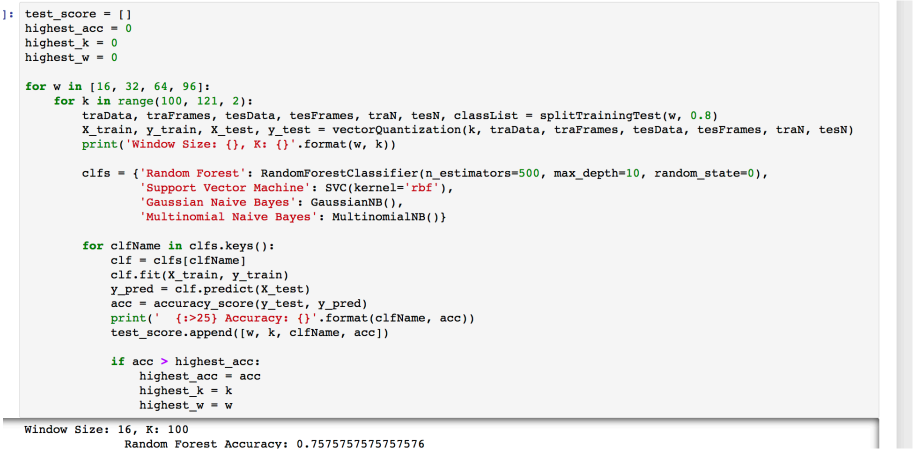
```
def createY(N, Y):
    classValue = 0
    for i in range(len(N)):
        for j in range(N[i]):
            Y.append(classValue)
        classValue += 1
    return np.array(Y).reshape(-1,)
```



```
def vectorQuantization(k, traData, traFrames, tesData, tesFrames, traN, tesN):
    kMean = KMeans(n_clusters = k)
    kMean.fit(traData)
    cluster = kMean.labels_
    predict = kMean.predict(tesData)
    y_train = []
    y_test = []
    X_train = createX(traFrames, cluster, k)
    y_train = createY(traN, y_train)
    X_test = createX(tesFrames, predict, k)
    y_test = createY(tesN, y_test)
    return X_train, y_train, X_test, y_test
```

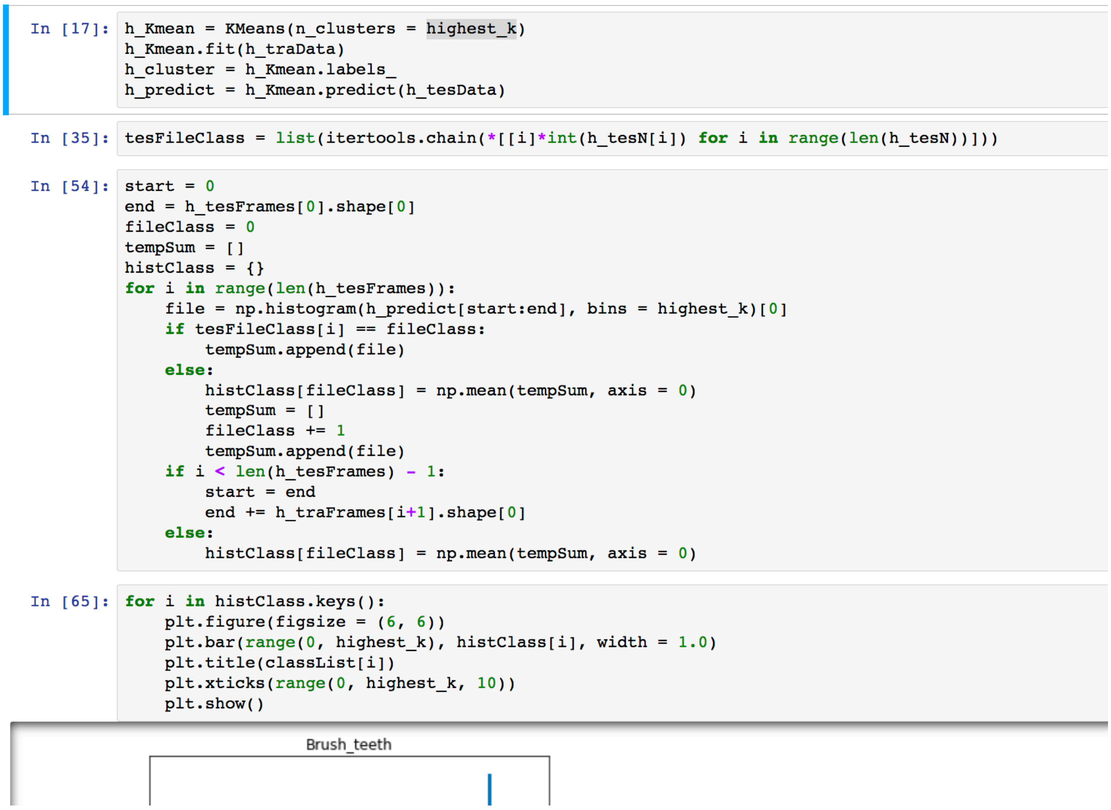


```
def plotConfusionMatrix(y_true, y_pred, classList, clfName):
    cm=confusion_matrix(y_true, y_pred)
    fig=plt.figure(figsize=(9, 9))
    plt.clf()
    plt.imshow(cm, interpolation='nearest', cmap='Reds')
    plt.title('Confusion Matrix - {}'.format(clfName))
    plt.ylabel('True label',fontsize=15)
    plt.xlabel('Predicted label',fontsize=15)
    tick_marks = range(len(classList))
    plt.xticks(tick_marks, classList, rotation=45, fontsize=12)
    plt.yticks(tick_marks, classList, fontsize=12)
    plt.show()
```



```
]: test_score = []
   highest_acc = 0
   highest_k = 0
   highest_w = 0

   for w in [16, 32, 64, 96]:
       for k in range(100, 121, 2):
           traData, traFrames, tesData, tesFrames, traN, tesN, classList = splitTrainingTest(w, 0.8)
           X_train, y_train, X_test, y_test = vectorQuantization(k, traData, traFrames, tesData, tesFrames, traN, tesN)
           print('Window Size: {}, K: {}'.format(w, k))

           clfs = {'Random Forest': RandomForestClassifier(n_estimators=500, max_depth=10, random_state=0),
                   'Support Vector Machine': SVC(kernel='rbf'),
                   'Gaussian Naive Bayes': GaussianNB(),
                   'Multinomial Naive Bayes': MultinomialNB()}

           for clfName in clfs.keys():
               clf = clfs[clfName]
               clf.fit(X_train, y_train)
               y_pred = clf.predict(X_test)
               acc = accuracy_score(y_test, y_pred)
               print('  {:>25} Accuracy: {}'.format(clfName, acc))
               test_score.append([w, k, clfName, acc])

               if acc > highest_acc:
                   highest_acc = acc
                   highest_k = k
                   highest_w = w
```

```
Window Size: 16, K: 100
              Random Forest Accuracy: 0.7575757575757576
```



```
In [17]: h_Kmean = KMeans(n_clusters = highest_k)
         h_Kmean.fit(h_traData)
         h_cluster = h_Kmean.labels_
         h_predict = h_Kmean.predict(h_tesData)
```

```
In [35]: tesFileClass = list(itertools.chain(*[[i]*int(h_tesN[i]) for i in range(len(h_tesN))]))
```

```
In [54]: start = 0
         end = h_tesFrames[0].shape[0]
         fileClass = 0
         tempSum = []
         histClass = {}
         for i in range(len(h_tesFrames)):
             file = np.histogram(h_predict[start:end], bins = highest_k)[0]
             if tesFileClass[i] == fileClass:
                 tempSum.append(file)
             else:
                 histClass[fileClass] = np.mean(tempSum, axis = 0)
                 tempSum = []
                 fileClass += 1
                 tempSum.append(file)
             if i < len(h_tesFrames) - 1:
                 start = end
                 end += h_traFrames[i+1].shape[0]
             else:
                 histClass[fileClass] = np.mean(tempSum, axis = 0)
```

```
In [65]: for i in histClass.keys():
             plt.figure(figsize = (6, 6))
             plt.bar(range(0, highest_k), histClass[i], width = 1.0)
             plt.title(classList[i])
             plt.xticks(range(0, highest_k, 10))
             plt.show()
```