CS498 AMO Homework 5

Team :
Minyuan Gu (minyuan3@illinois.edu, netid minyuan3)
Yanislav Shterev (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:

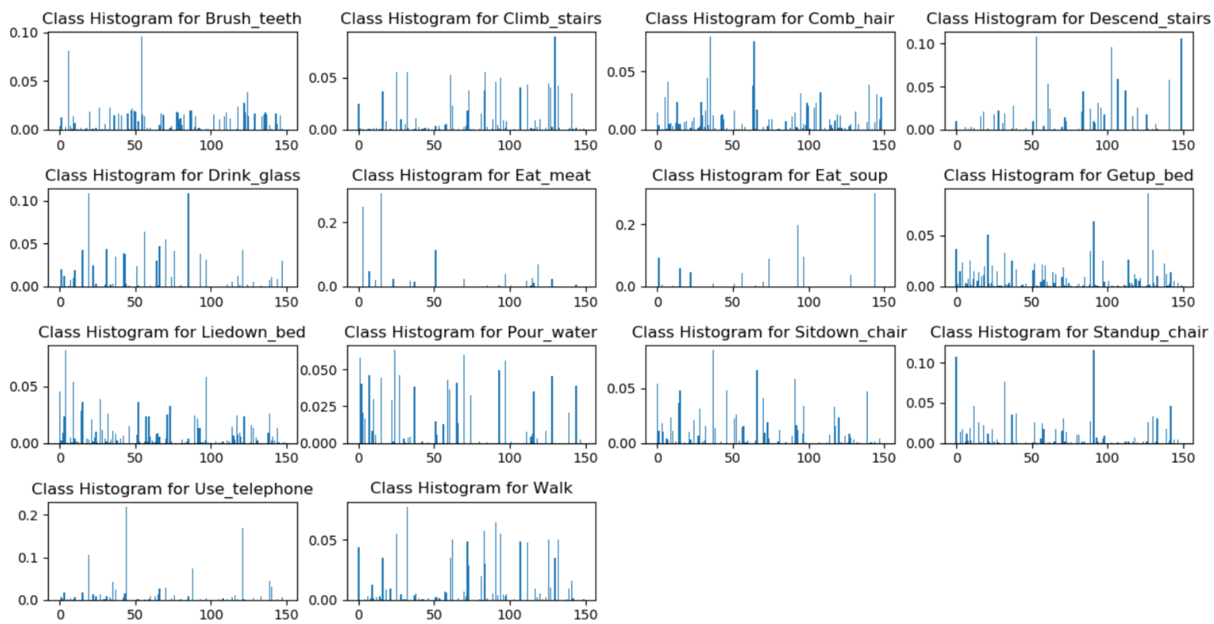
1. We tried different segment sizes (16 – 48) and they all have similar accuracy and size 32 and 48 were chosen since they provided good results and serves as a sliding window of 1~1.5 seconds of the information from activities.
2. Overlap X% has big influence on the accuracy. Our code introduced parameters (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.

Fixed length size	Overlap (0-X%)	K-value	Kmean type	Classifier	Averaged_Accuracy
4	70%	40	standard	RandomForest (30Trees 16 Depth)	76.842%
10	70%	40	standard	RandomForest (30Trees 16 Depth)	75.789%
16	0%	100	standard	RandomForest (30Trees 16 Depth)	71.228%
16	70%	600	Hierarchical-Kmean	RandomForest (30Trees 16 Depth)	78.480%
32	0%	120	standard	RandomForest (30Trees 16 Depth)	69.825%
32	0%	200	standard	RandomForest (30Trees 16 Depth)	71.930%
32	30.00%	100	standard	RandomForest (30Trees 16 Depth)	74.269%
32	70%	200	standard	RandomForest (30Trees 16 Depth)	78.480%
32	70%	480	Hierarchical-Kmean	RandomForest (30Trees 16 Depth)	78.246%
32	70%	800	Hierarchical-Kmean	RandomForest (30Trees 16 Depth)	74.269%
48	30.00%	100	standard	RandomForest (30Trees 16 Depth)	75.556%
48	70.00%	150	standard	RandomForest (30Trees 16 Depth)	81.988%
64	0%	120	standard	RandomForest (30Trees 16 Depth)	68.070%
64	30.00%	100	standard	RandomForest (30Trees 16 Depth)	71.462%
32	50.00%	120	standard	RandomForest (120Trees 30 Depth)	81.301%
32	70.00%	110	standard	RandomForest (120Trees 30 Depth)	81.520%
32	90.00%	100	standard	RandomForest (120Trees 30 Depth)	83.664%
32	90.00%	500	Hierarchical-Kmean	RandomForest (120Trees 30 Depth)	82.143%
48	50.00%	100	standard	RandomForest (120Trees 30 Depth)	80.573%
48	90.00%	110	standard	RandomForest (120Trees 30 Depth)	84.169%
48	90.00%	500	Hierarchical-Kmean	RandomForest (120Trees 30 Depth)	82.247%
48	90.00%	150	standard	RandomForest (120Trees 30 Depth)	83.876%
48	90.00%	300	standard	RandomForest (120Trees 30 Depth)	83.772%
32	70.00%	100	standard	KNeighbors (1 neigbor 30 leafs)	71.285%
48	70.00%	120	standard	KNeighbors (1 neigbor 30 leafs)	71.277%

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.

	Brush_teeth	Climb_stairs	Comb_hair	Descend_stair	Drink_glass	Eat_meat	Eat_soup	Getup_bed	Liedown_bed	Pour_water	Sitdown_chair	Standup_chair	Use_telephon	Walk	Accuracy
Brush_teeth	3	0	0	0	0	0	0	0	0	0	1	0	0	0	75.00%
Climb_stairs	0	30	0	0	0	0	0	0	0	0	0	0	0	4	88.24%
Comb_hair	0	0	10	0	1	0	0	0	0	0	0	0	0	0	90.91%
Descend_stairs	0	3	0	10	0	0	0	0	0	0	0	0	0	1	71.43%
Drink_glass	0	0	0	0	34	0	0	0	0	0	0	0	0	0	100.00%
Eat_meat	0	0	0	0	0	1	0	0	0	1	0	0	0	0	50.00%
Eat_soup	0	0	0	0	0	0	1	0	0	0	0	0	0	0	100.00%
Getup_bed	0	0	0	0	0	0	0	29	1	0	0	4	0	0	85.29%
Liedown_bed	0	0	0	0	0	0	0	1	0	1	7	0	0	1	0.00%
Pour_water	0	0	0	0	0	0	0	0	0	33	1	0	0	0	97.06%
Sitdown_chair	0	0	0	0	0	0	0	0	0	0	34	0	0	0	100.00%
Standup_chair	0	0	0	0	0	0	0	0	0	0	1	33	0	0	97.06%
Use_telephone	0	0	0	0	2	0	0	0	0	0	1	0	2	0	40.00%
Walk	0	1	0	0	0	0	0	0	0	0	0	3	0	30	88.24%

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.
Also please zoom in the pdf, otherwise we notice all the underscores ('_') are missing or displayed improperly in pdf!

i) 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
```

ii) 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
```

iii) Generating the histogram

Two steps to generate histogram:
1. Build dictionary (split sequence + Kmean)
2. Use dictionary to quantize vector by building histograms.

```
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)
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)
```

iv) 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