Concrete Machine Learning

Deep User : 2020 Summer Program

K – means Clustering

Clustering algorithm

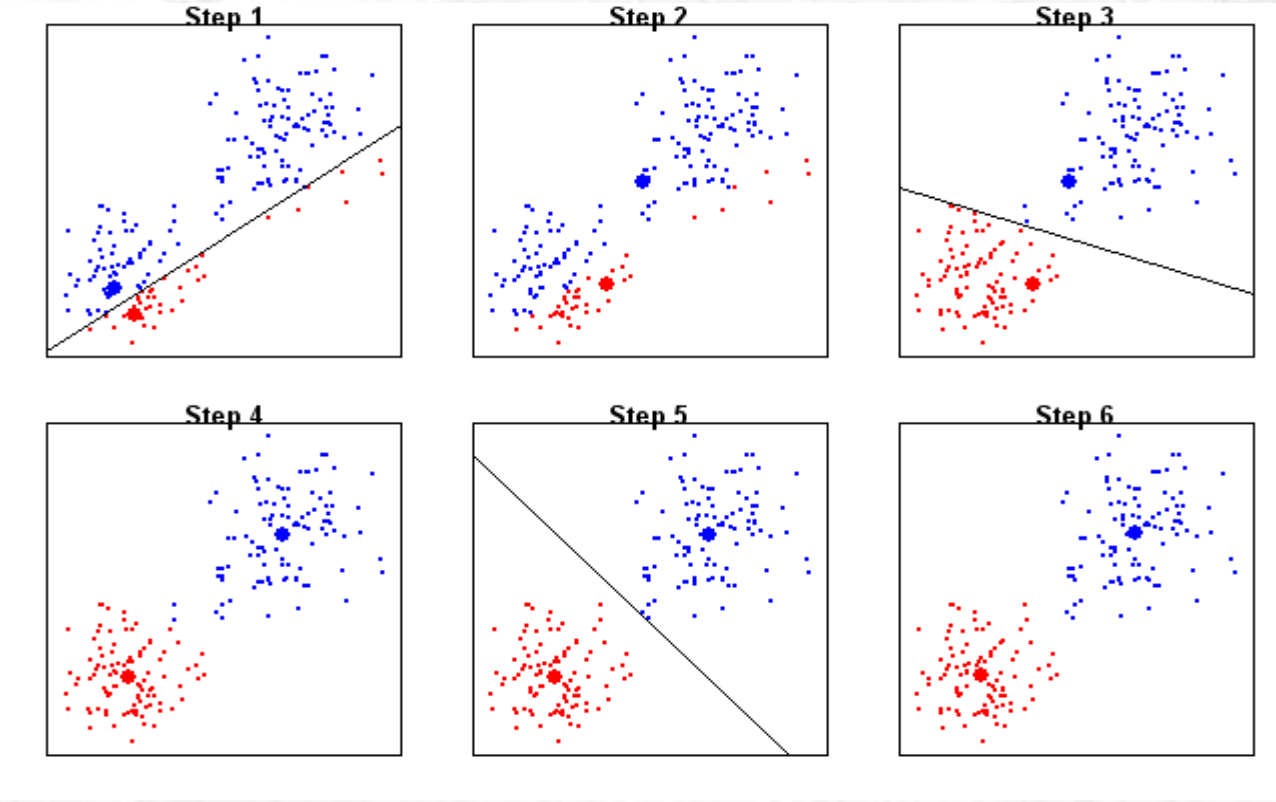
Unsupervised learning

Easy & Simple

Labeling process

Get internal structure information

Knowledge discovery in data



A | K-Means

Main Purpose

$$minimize_{c1,\ldots,ck} \sum_{k=1}^{k} W(C_k)$$

$W(C_k)$ = Distribution within the cluster of the kth cluster – avg of kth clusters sum

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

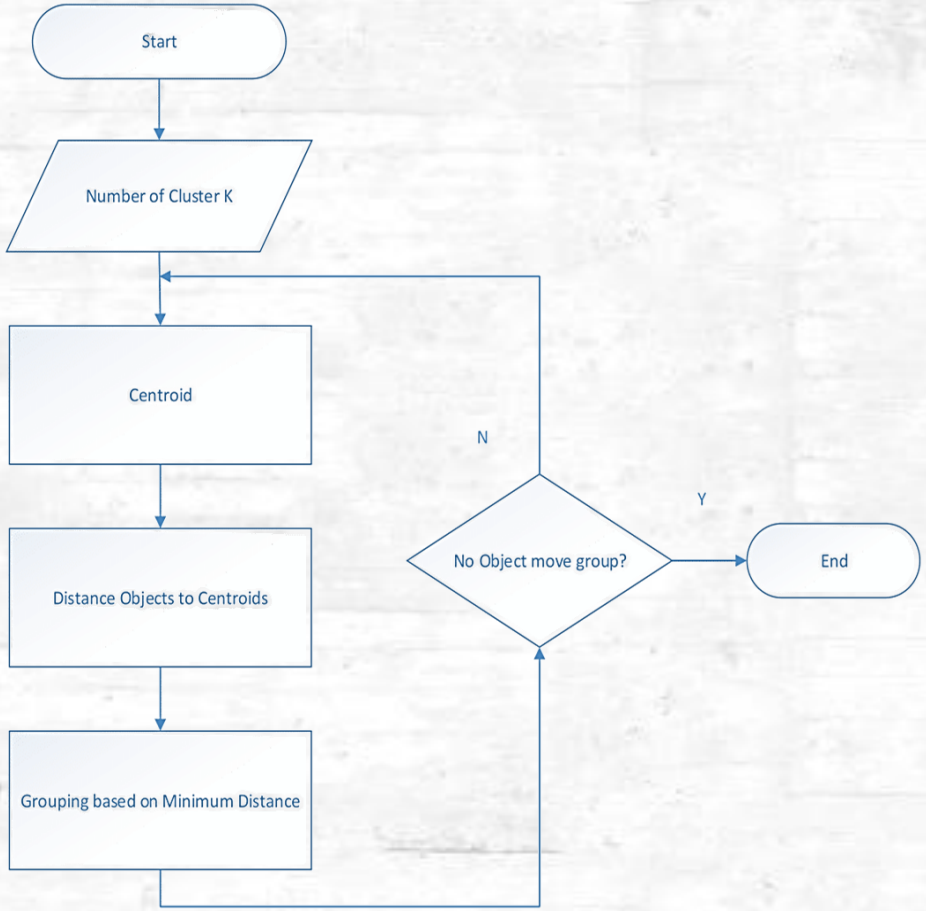
* Use as a cost function

Repeat until $W(C_k)$ is convergence condition is satisfied.

A | K-Means

1. Pick an arbitrary center value μk (‘k’ value means number of clusters that want to divide).
2. Calculate the ***distance** from the center to each data.
3. Update the cluster by selecting the closest center in each data.
4. Recalculate the center for the recreated clusters and repeat steps 1 to 4 until the ****convergence** condition is satisfied

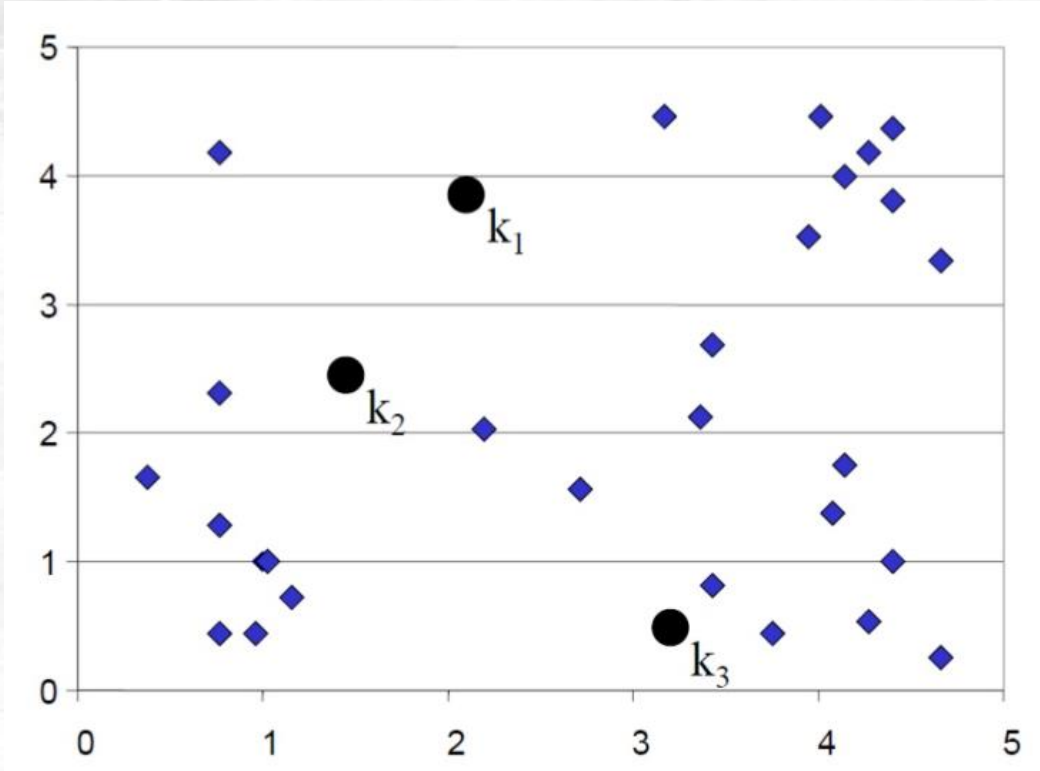
Solved through repetitive procedures



* Means the center of the cluster is not update during the process

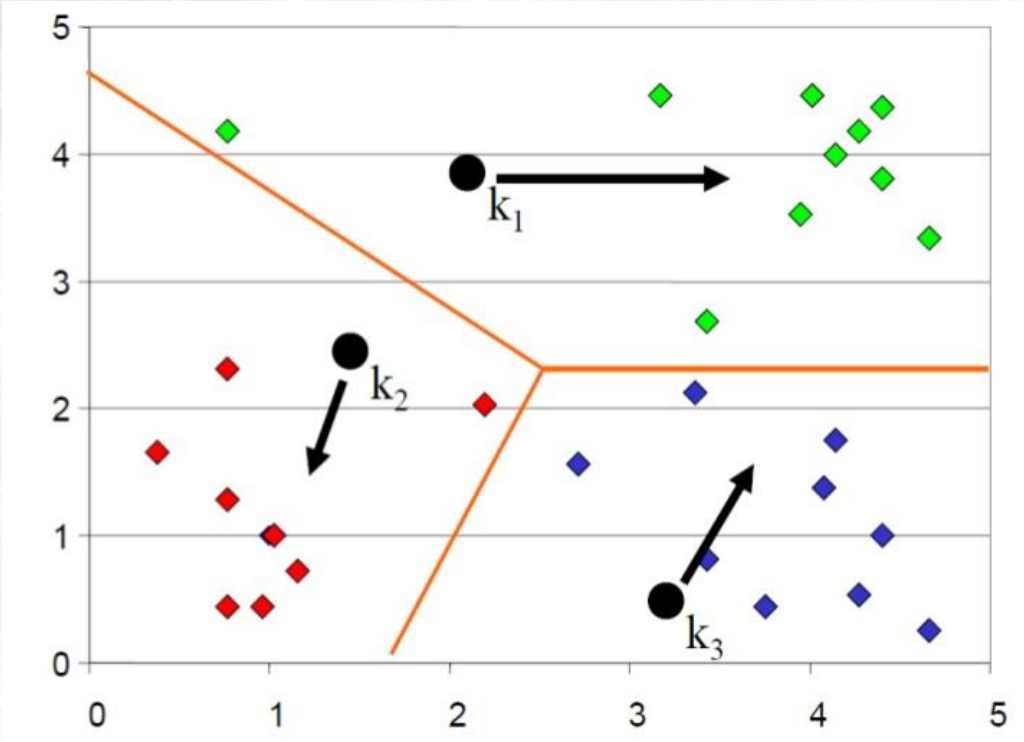
** Distance means square of euclidean distance

A | K-Means



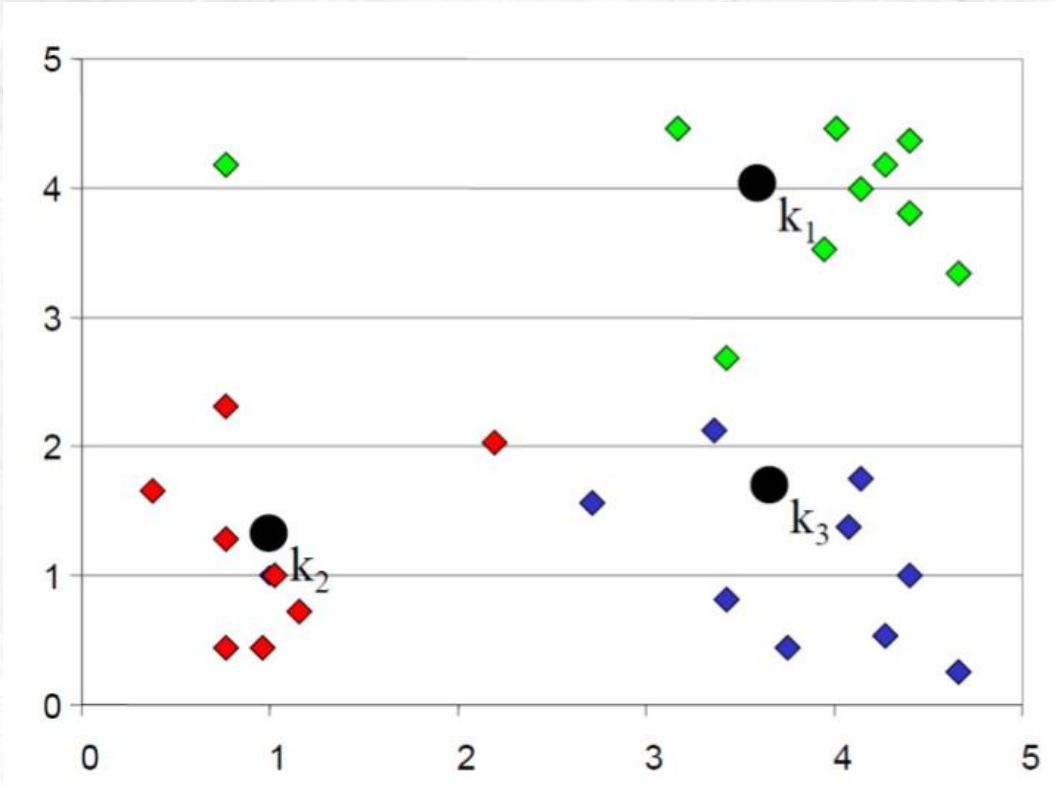
① Pick an first ‘k(number of clusters)’ arbitrary center

A | K-Means



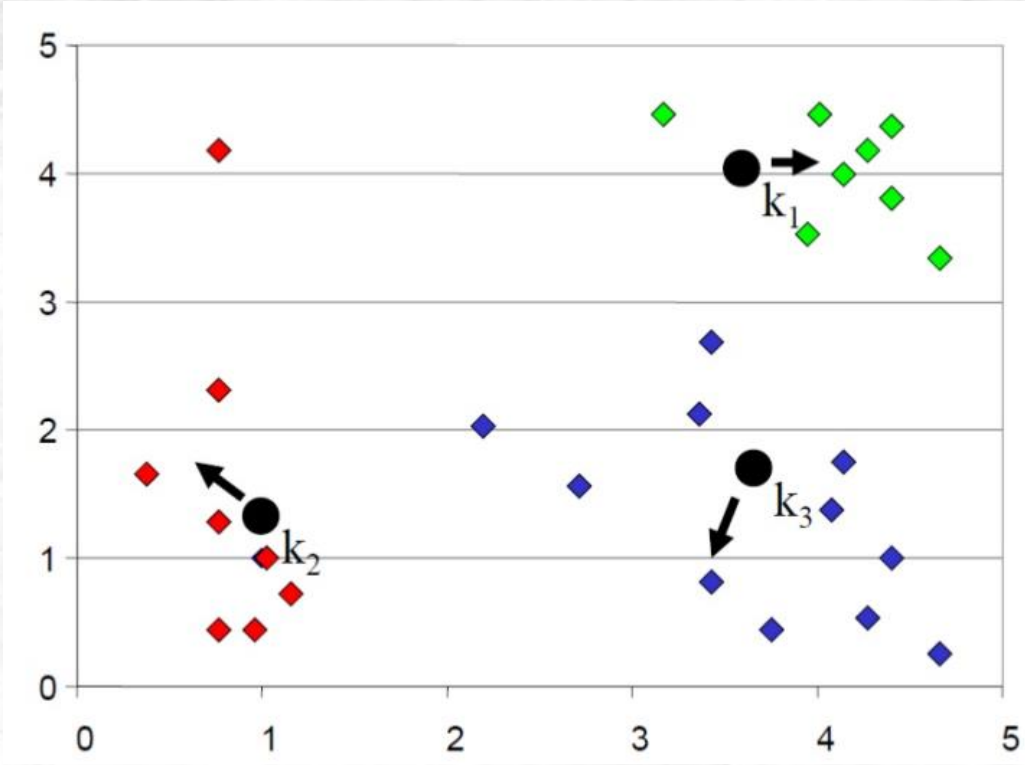
① Assign each object to the closest centroid k_i.
② Averaging objects that assigned same centroid.

A | K-Means



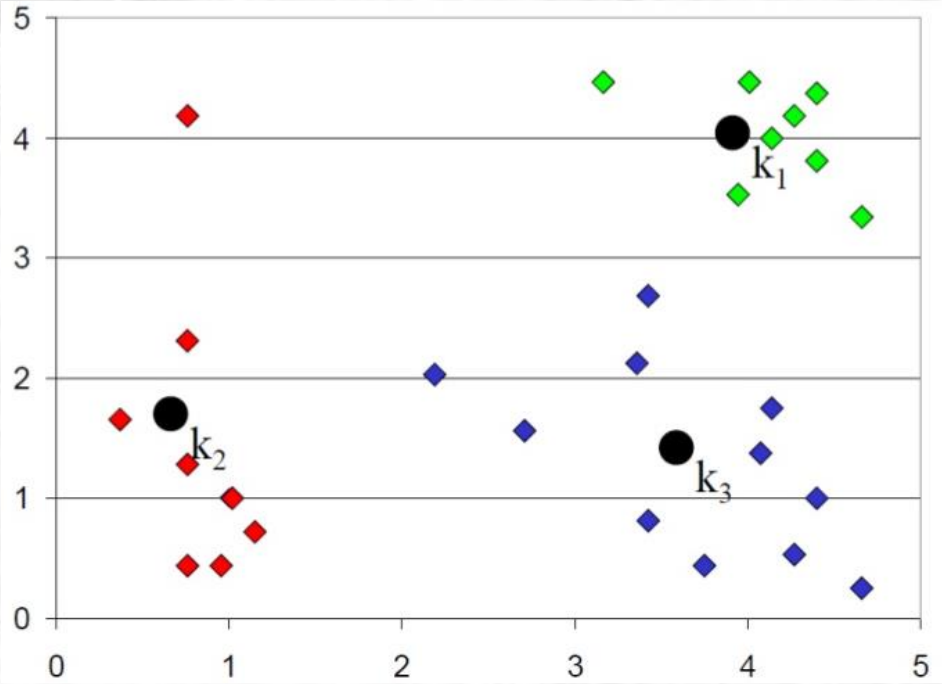
① Use calculated average value as a 2nd centroid

A | K-Means



① Assign each object to the closest centroid k_i.(2nd centroid)
② Averaging objects that assigned same centroid.

A | K-Means

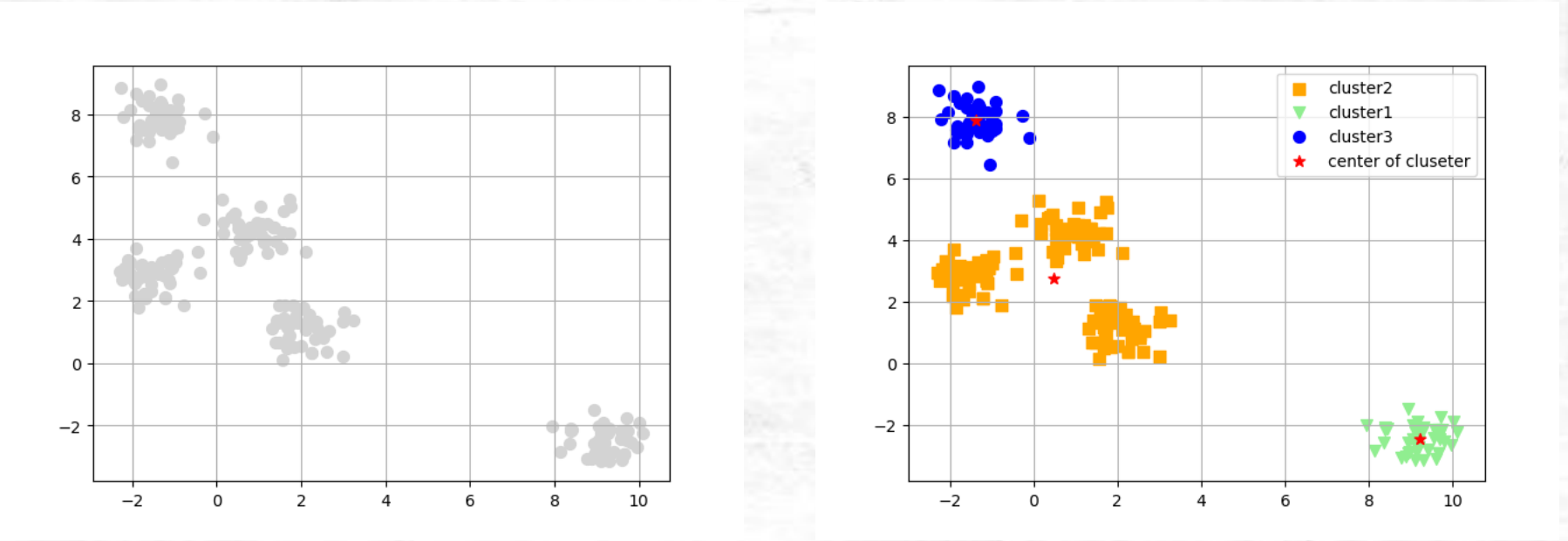


① Use calculated average value as a 3rd centroid

Repeat until there is no change in the membership of the objects

A | K-Means

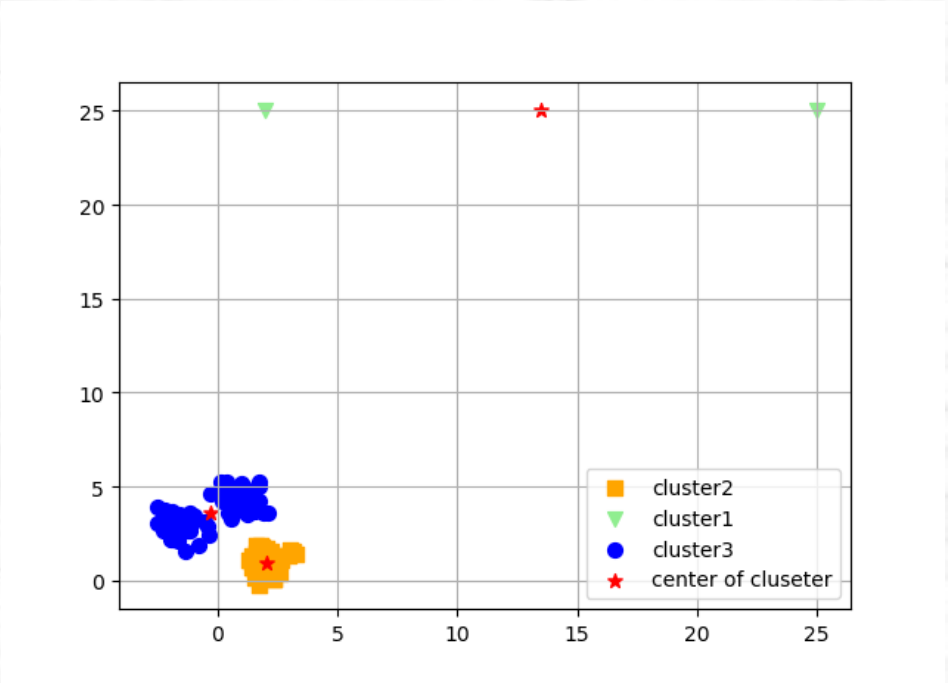
① The number of clusters k must be specified as an input parameter



If the K value is significantly different from the actual number of meaningful clusters in the dataset, the result may not be good.

A | K-Means

② Sensitive to outliers

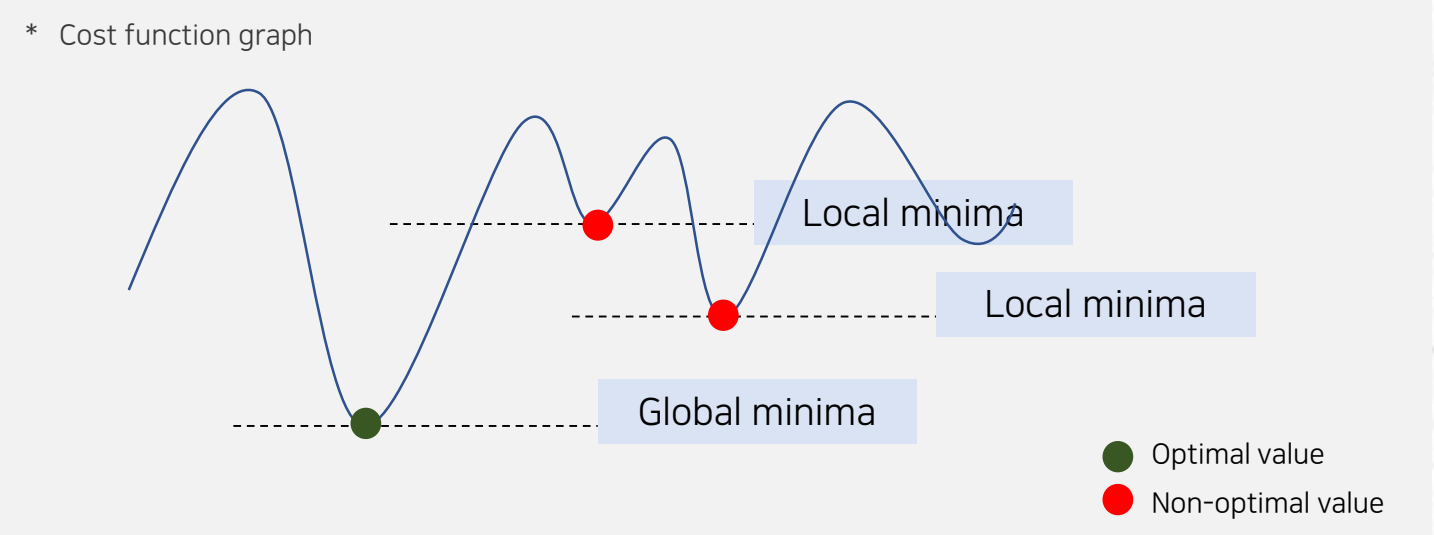


In the process of updating the center point, an outliers value can greatly distort the overall average value in the cluster

③ Easy to fall into local minimum

Repeat until there is **no change in the membership of the objects**

* Cost function graph



The convergence condition of the algorithm is satisfied even if the local minimum value is reached instead of the global, so that the optimization is no longer advanced

A | K-Means

① The number of clusters k must be specified as an input parameter

Solve with sequential algorithm

② Sensitive to outliers

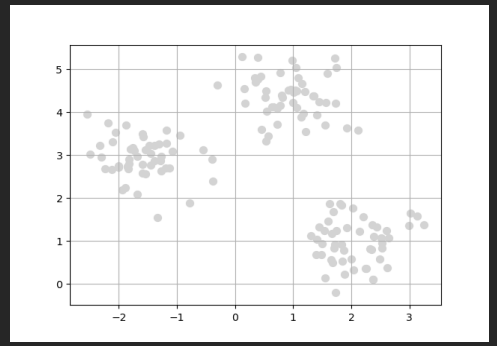
Select representative samples from the samples and calculate them.
Excluding outlier values through Preprocessing.

③ Easy to fall into local minimum

Repeat several times, changing the initial value.
Use smallest cost function among the repeated values is selected.

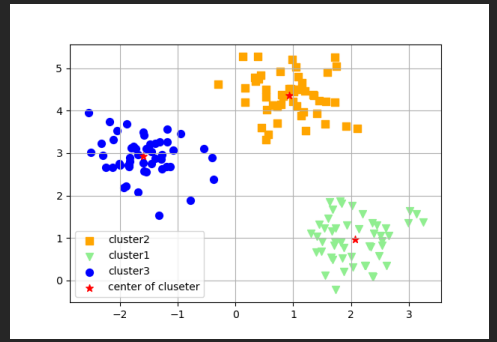
A | K-Means

```
...
X,y = make_blobs(n_samples=150, n_features=2, centers=3, cluster_std=0.5, shuffle=True, random_state=0)
#Make random samples

plt.scatter(X[:, 0],X[:, 1],c='lightgray',marker='o',s=50)
plt.grid(True)
plt.show()

init_centroid='random'

km = KMeans(n_clusters = 3, init = init_centroid, random_state=0)    # Make 3cluster group, init μk as random
y_km = km.fit_predict(X)     #Get K-means clustered data (numpy array data)

plt.scatter(X[y_km == 0,0],X[y_km == 0,1], c='orange',marker='s',s=50, label='cluster2')
plt.scatter(X[y_km == 1,0],X[y_km == 1,1], c='lightgreen',marker='v',s=50, label='cluster1')
plt.scatter(X[y_km == 2,0],X[y_km == 2,1], c='blue',marker='o',s=50, label='cluster3')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],c='red', marker='*',s=50, label='center of cluseter')

...
```



A | K-Means

```
...
X,y = make_blobs(n_samples=150, n_features=2, centers=3, cluster_std=0.5, shuffle=True, random_state=0)
#Make random samples

plt.scatter(X[:, 0],X[:, 1],c='lightgray',marker='o',s=50)
plt.grid(True)
plt.show()

init_centroid='random'

km = KMeans(n_clusters = 3, init = init_centroid, random_state=0)    # Make 3cluster group, init μk as random
y_km = km.fit_predict(X)     #Get K-means clustered data (numpy array data)

plt.scatter(X[y_km == 0,0],X[y_km == 0,1], c='orange',marker='s',s=50, label='cluster2')
plt.scatter(X[y_km == 1,0],X[y_km == 1,1], c='lightgreen',marker='v',s=50, label='cluster1')
plt.scatter(X[y_km == 2,0],X[y_km == 2,1], c='blue',marker='o',s=50, label='cluster3')
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],c='red', marker='*',s=50, label='center of cluseter')
...
```



$$X = C_1 \cup C_2 \ldots \cup C_K, \quad C_i \cap C_j = \phi$$

$$argmin_C \sum_{i=1}^{K} \sum_{x_j \in C_i} \|x_j - c_i\|^2$$

❶ Assign member based on current centers.

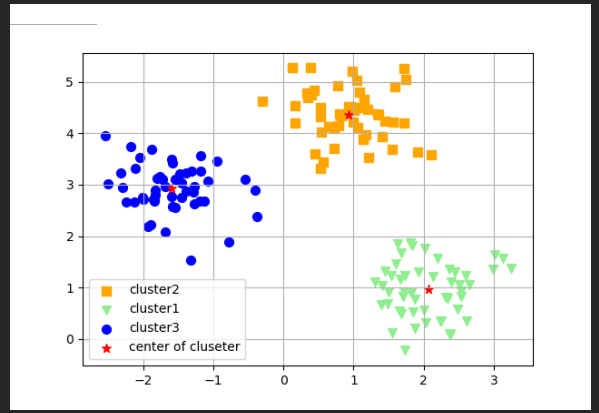
❷ Re-estimate centers based on current assignment.

A | K-Means

```python
km = KMeans(n_clusters = 3, init = init_centroid, random_state=0)    # Make 3cluster group, init μk as random
y_km = km.fit_predict(X)     #Get K-means clustered data (numpy array data)
```

```
...
while (loop) { //when the k-positions are all same with next position.
               //center init

    for (int i = 0; i<K_COUNT; i++) {
        center[i].x = 0;
        center[i].y = 0;
        count_Group[i] = 0;
    }

    // distance
    for (int i = 0; i<datas.size(); i++) {
        for (int j = 0; j<K_COUNT; j++) {
            double tmp_distance = sqrt(pow(k[j].x - datas[i].x, 2) +
                pow(k[j].y - datas[i].y, 2));
                distance[j][i] = tmp_distance;
        }
    }



    //get center
    for (int i = 0; i<datas.size(); i++) {
        double min = distance[0][i];
        int min_j = 0;

        for (int j = 1; j<K_COUNT; j++) {
            if (min > distance[j][i]) {
                min = distance[j][i];
                min_j = j;
            }
        }
        center[min_j].x += datas[i].x;
        center[min_j].y += datas[i].y;
        count_Group[min_j]++;
    }
```

```
    //change K

    int same_count = 0;

    for (int i = 0; i<K_COUNT; i++) {
        if (count_Group[i] != 0) {
            if ((center[i].x / count_Group[i]) == k[i].x
                && (center[i].y / count_Group[i] == k[i].y))
                same_count++;
            k[i].x = center[i].x / count_Group[i];
            k[i].y = center[i].y / count_Group[i];
        }

        if (same_count == K_COUNT) {
            loop = false;
        }
        cout << fixed << setprecision(2);
        cout << "(" << k[i].x << "," << k[i].y << ") ";
    }cout << endl;
}//end of loop
...
```



A | K-Means

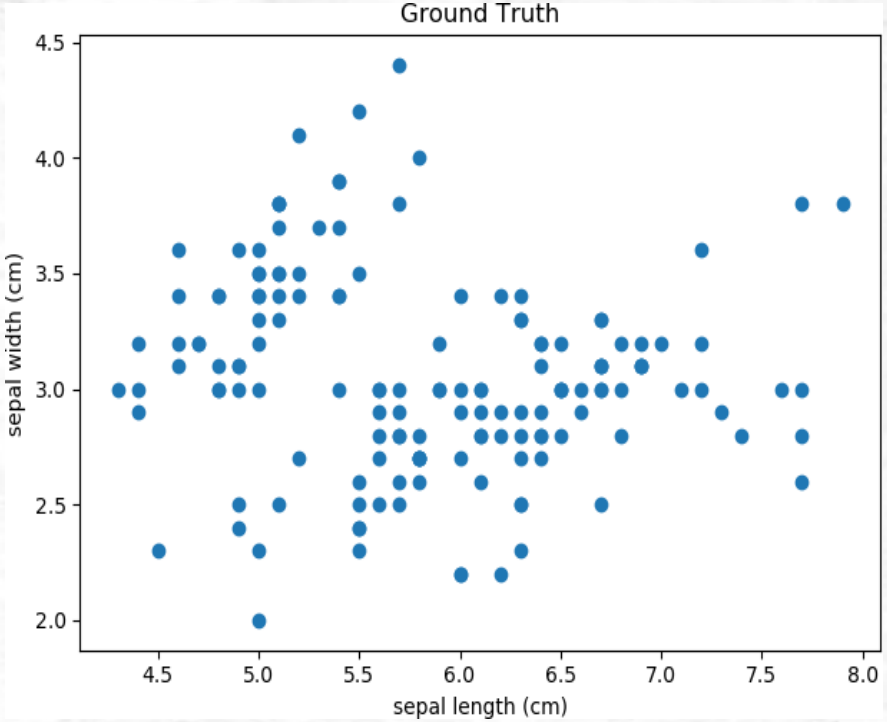
K-means clustering step (N = data point)

Algorithm 1: K-means clustering

Input : a given data $X = \{x_1, x_2, \ldots, x_N\}$
the number of clusters K,
maximum number of iteration I

Output : clustering results r_{nk} for all n and k,
centroid of clusters C

Randomly initialize $C = \{c_1, c_2, \ldots, c_k\}$
for $t = 1 : I$ **do**
  // Assignment step
  for $n = 1 : N$ **do**
    $$r_{nk} = \begin{cases} 1, & \text{if } k = \underset{i}{\operatorname{argmin}} ||x_n - c_i||^2 \\ 0, & \text{otherwise} \end{cases}$$
  end
  // Update step
  for $k = 1 : K$ **do**
    $$c_k = \frac{1}{\sum_{n=1}^{N} r_{nk}} \sum_{n=1}^{N} r_{nk} x_n$$
  end
end



Data feature

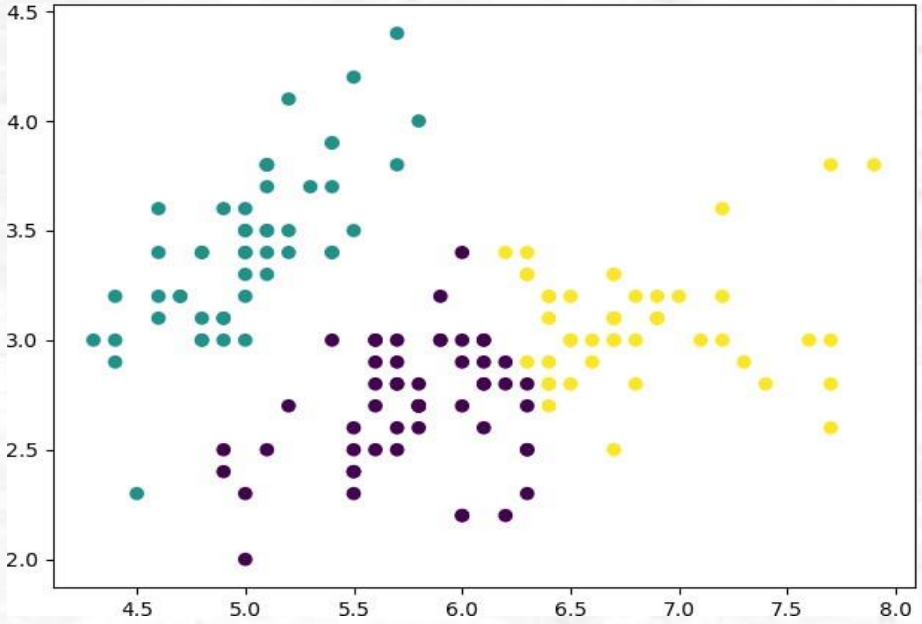
Sepal width, Sepal length

A | K-Means

Example result of Kmeans sklearn library

After build all of function, you can see below result from python console when you compile “main.py”

Use the scatter color which do you want.



A | K-Means

```
def __init__(self, k, data, iteration): # initalize
    self.k = k # number of cluster
    self.data = data    # data
    self.iteration = iteration # set iteration [300]

def Centroids(self, ):  # Set initali centorids
    data = self.data.to_numpy() # get data and change numpy for sampling
    idx = np.random.randint(int(np.size(data,0)), size=int(self.k)) # get random index by using randint
    sampled_cen = data[idx,:] # sampling...
    return sampled_cen #return init centers
```

A | K-Means

```
def get_UD(self,data, centroids, clusters): # Get Uclideint distance
    for ins in data:  #for whole data
        mu = min([(i[0], np.linalg.norm(ins-centroids[i[0]])) \
                            for i in enumerate(centroids)], key=lambda t:t[1])[0] # Get uclidient distnace formula
        try: # exception processing
            clusters[mu].append(ins) #for all clusters append instance (as a asssignment functino)
        except KeyError: # exceptino handling
            clusters[mu] = [ins] #update case
    for result in clusters: #for all sub-clusters
        if not result: #Nan case
            result.append(data[np.random.randint(0, len(data), size=1)].flatten().tolist()) # Samplilng and apeend sub-clusters
    return clusters     # return whole clustsers k sub-clusters
```

A | K-Means

```
def Assignment(self, ): # code for overall process
    data = self.data.to_numpy() # change data to numpy for future processing
    cen = self.Centroids() # Get initial centroids
    prev_centro = [[] for i in range(self.k)] # get prev_cnetroid for comparing
    iters = 0 # set intertionflag 0
    warnings.simplefilter(action="ignore", category=FutureWarning) #ignore warnings check return part of Update
    while self.Update(cen, prev_centro, iters) is not True: # cheking Update (for LIMITION)
        iters =  iters + 1 # update iteration counter
        clusters = [[] for i in range(self.k)] # Set cluster
        old_result = [[] for i in range(self.k)] # Set prev_cluster for comparing
        clusters = self.get_UD(data, cen, clusters) # Set cluster (Part of Assignment function)
        idx = 0 # Set index
        for result in clusters: # for whole clusters
            prev_centro[idx] = cen[idx] # update centroids
            cen[idx] = np.mean(result, axis=0).tolist() # Get center mean
            idx = idx+1 # update index counter
        if np.array_equal(old_result,result) is True:   # Comparing
            iters = 0 # if iteration is not same update iters to 0 (start from ground again)
        iteration = self.iteration # get iteration
        old_result = result # update result
    return clusters , iteration #return clustrrs and iterations
```

A | K-Means

```
def Update(self,centroids, prev_centro, iters): # Update as a teration checker and centroid assigmenrtor
    if iters > LIMITION: # compare for LIMITION (early stopping)
        return True # for let loop Assignmnet functino
    warnings.simplefilter(action='ignore', category=FutureWarning) #ignore warnings check return part of Update
    return prev_centro == centroids # Allocation
```

```
def Train(self, ):  # Train for get result and Processing overall kmeans workings
    itertaion = 0 # set interation 0 (init)
    result,iteration = self.Assignment() # get result and iteration
    self.iteration = itertaion # update iteration
    return result # return result
```

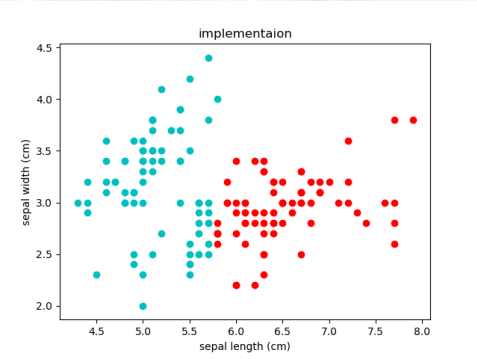
A | K-Means

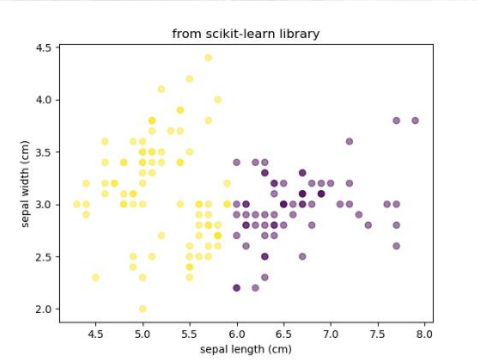
```
if __name__ == '__main__': # Start from main
    colorlist = ['r','c','k','g','m','b','y'] # Set color list (set this pallet because white and yellow is hard to congize)
    data = pd.read_csv('data.csv') # load data
    model1 = kmeans_(k=3, data=data, iteration=iteration) # implemented model init setting
    clustsers = model1.Train() #set clusters
    result = [] #result list for set diff colors
    for i in range(int(model1.get_k())): # for k case
        result = np.array(clustsers[i]) # i control for reslut
        result_x = result[:,0] # Assign x
        result_y = result[:,1] # Assign y
        plt.scatter(result_x,result_y,c=str((colorlist[i]))) #plt scatter for each clusters
    plt.xlabel('sepal length (cm)') # set label
    plt.ylabel('sepal width (cm)') # set label
    plt.title("implementaion") # set title
    plt.show() # show plot
```

A | K-Means

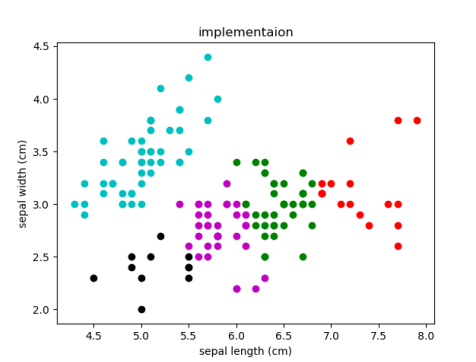
```
model2 = KMeans(n_clusters=3, init='random', random_state=seed_num, max_iter=iteration).fit(data) # sklearn model init setting
predict = pd.DataFrame(model2.predict(data)) # update predict label
predict.columns = ["predict"] # Set col name
data = pd.concat([data,predict],axis=1) # concat data
predict.columns=['predict'] # Set col name
plt.scatter(data['Sepal width'],data['Sepal length'],c=data['predict'],alpha=0.5) # scatter plot
plt.xlabel('sepal length (cm)') # set label
plt.ylabel('sepal width (cm)') # set label
plt.title("from scikit-learn library") # set title
plt.show() # show plot
```

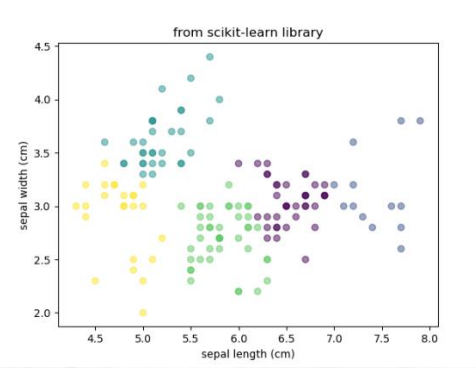
A | K-Means



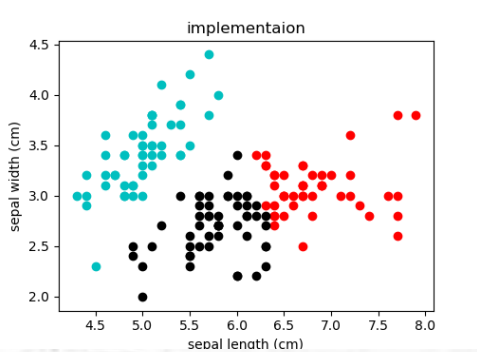


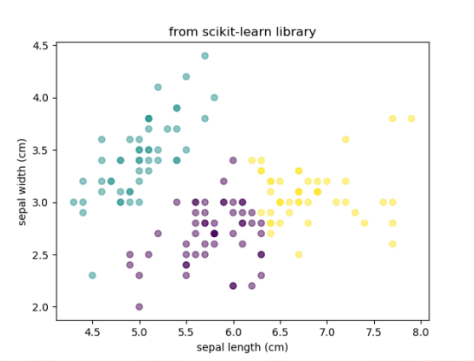
K = 2



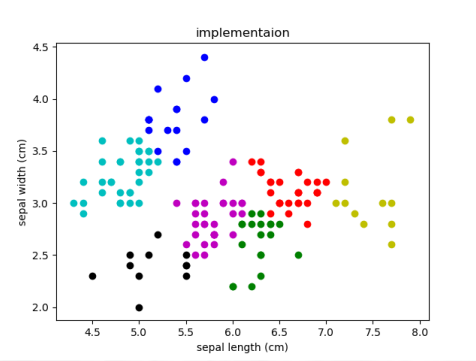


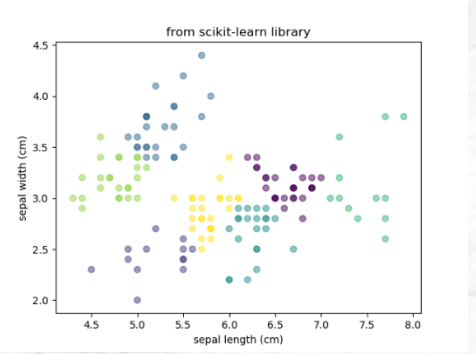
K = 5





K = 3





K = 7

DeepUser

K-means Algorithm

THANKS