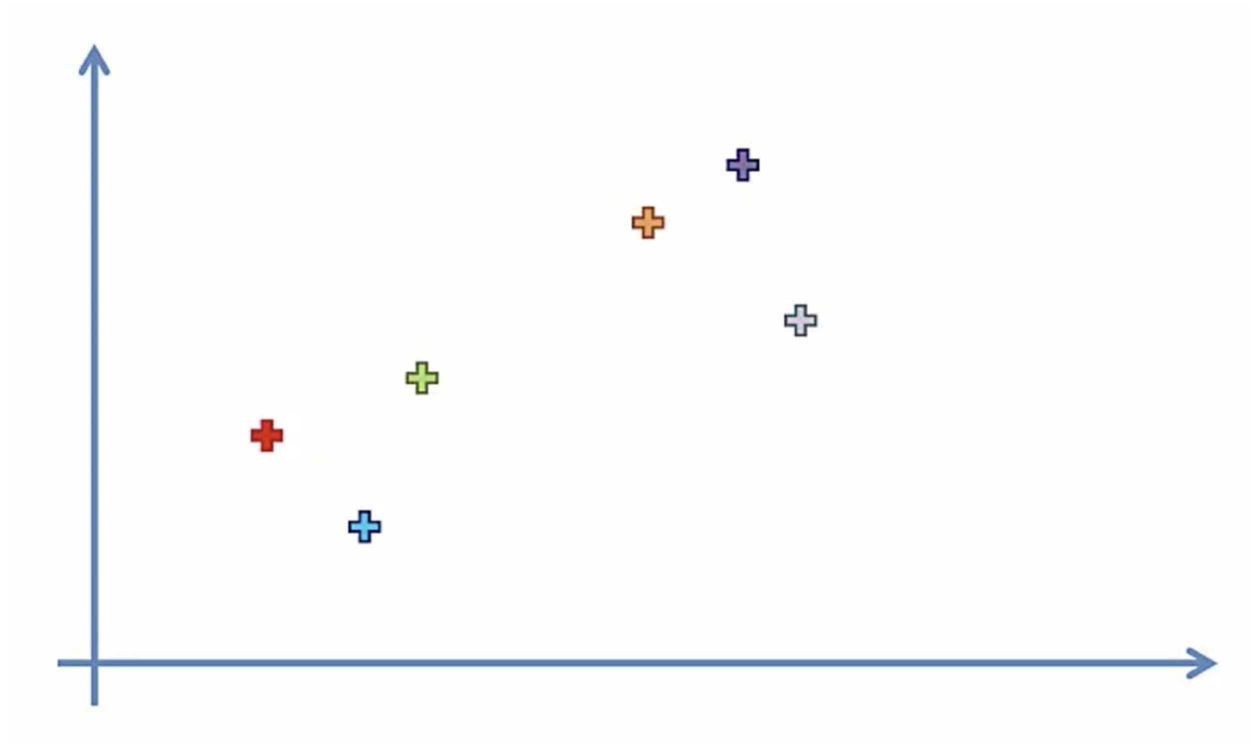
Theory of Hierarchical Clustering

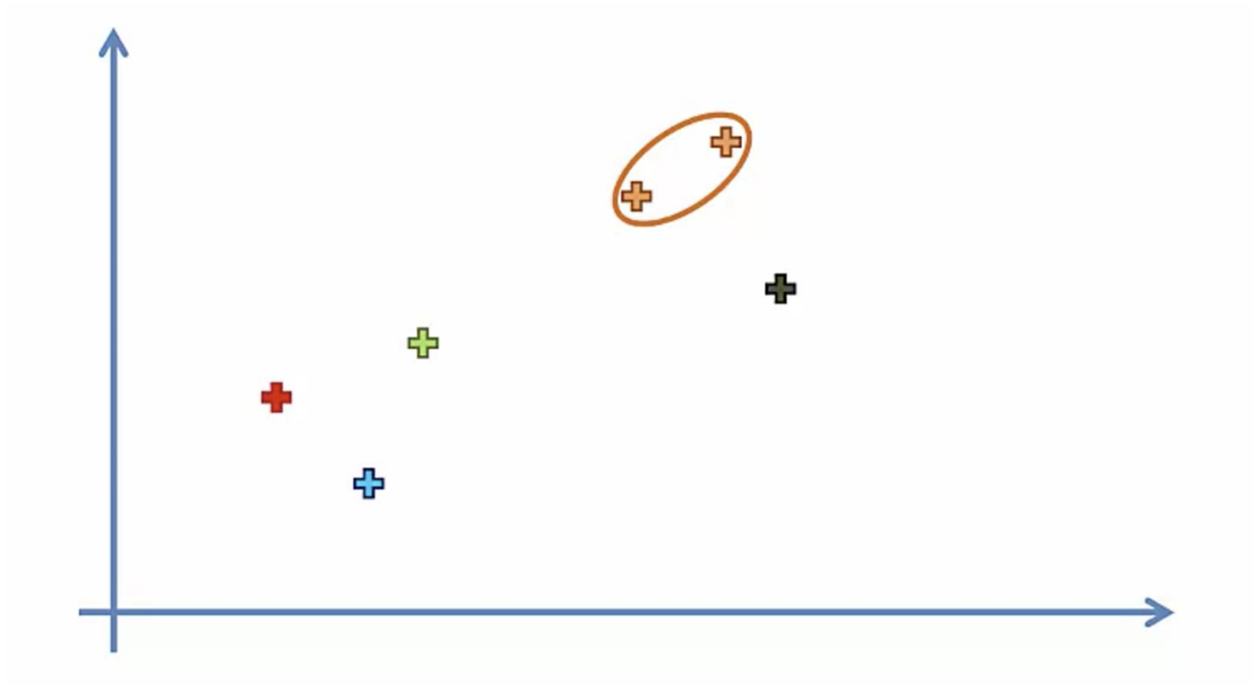
There are two types of hierarchical clustering: **Agglomerative** and **Divisive**. In the former, data points are clustered using a bottom-up approach starting with individual data points, while in the latter top-down approach is followed where all the data points are treated as one big cluster and the clustering process involves dividing the one big cluster into several small clusters.

Following are the steps involved in agglomerative clustering:

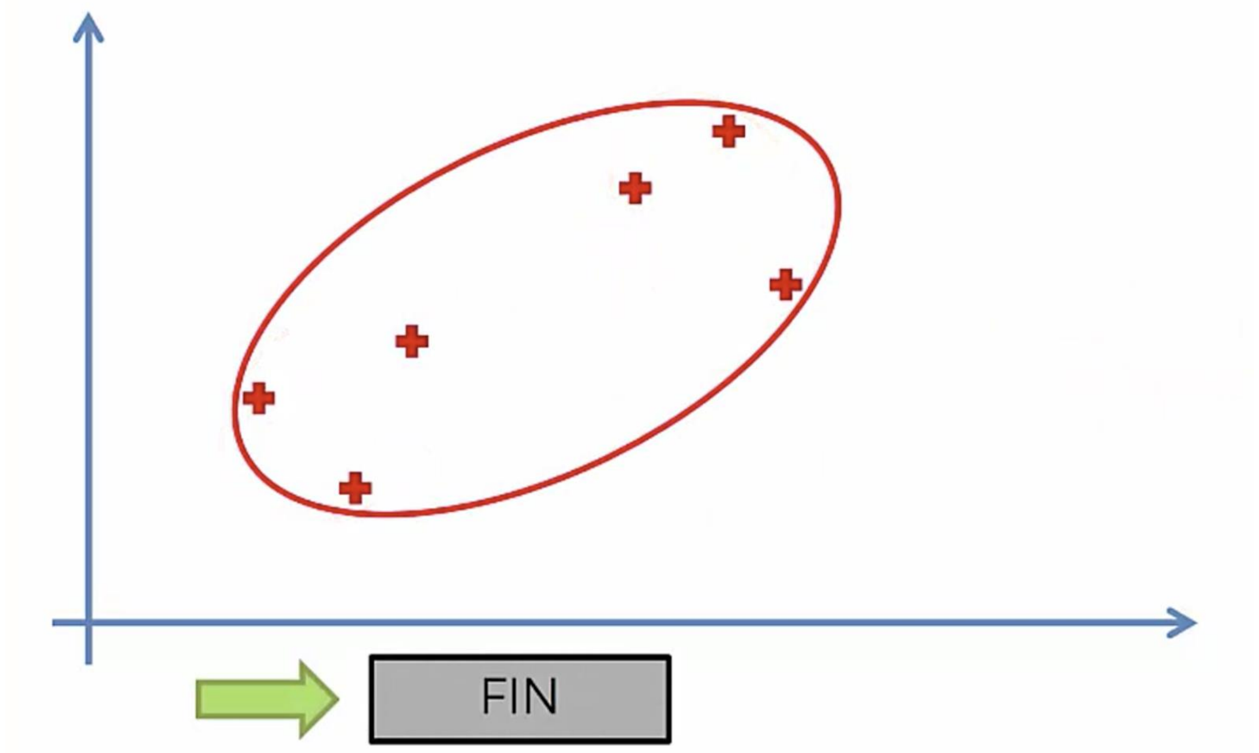
1. At the start, treat each data point as one cluster. Therefore, the number of clusters at the start will be K, while K is an integer representing the number of data points.



2. Form a cluster by joining the two closest data points resulting in K-1 clusters.



3. Repeat the above three steps until one big cluster is formed.



4.

5. Once single cluster is formed, dendrograms are used to divide into multiple clusters depending upon the problem. We will study the concept of dendrogram in detail in an upcoming section.

There are different ways to find distance between the clusters. The distance itself can be Euclidean or Manhattan distance. Following are some of the options to measure distance between two clusters:

1. Measure the distance between the closes points of two clusters.
2. Measure the distance between the farthest points of two clusters.
3. Measure the distance between the centroids of two clusters.
4. Measure the distance between all possible combination of points between the two clusters and take the mean.

pdist

Pairwise distance between observations

Description

`Y = pdist(X)` computes the Euclidean distance between pairs of objects in *m*-by-*n* matrix `X`, which is treated as *m* vectors of size *n*. For a dataset made up of *m* objects, there are $(m-1)\cdot m/2$ pairs.

The output, `Y`, is a vector of length $(m-1)\cdot m/2$, containing the distance information. The distances are arranged in the order (1,2), (1,3), ..., (1,*m*), (2,3), ..., (2,*m*), ..., ..., (*m*-1,*m*). `Y` is also commonly known as a similarity matrix or dissimilarity matrix.

To save space and computation time, `Y` is formatted as a vector. However, you can convert this vector into a square matrix using the `squareform` function so that element *i*,*j* in the matrix, where $i<j$, corresponds to the distance between objects *i* and *j* in the original dataset.

`Y = pdist(X,'metric')` computes the distance between objects in the data matrix, `X`, using the method specified by `'metric'`, where `'metric'` can be any of the following character strings that identify ways to compute the distance.

`'euclidean'`	Euclidean distance (default)

`'seuclidean'`	Standardized Euclidean distance. Each coordinate in the sum of squares is inverse weighted by the sample variance of that coordinate.
`'mahalanobis'`	Mahalanobis distance
`'cityblock'`	City Block metric
`'minkowski'`	Minkowski metric
`'cosine'`	One minus the cosine of the included angle between points (treated as vectors)
`'correlation'`	One minus the sample correlation between points (treated as sequences of values).
`'hamming'`	Hamming distance, the percentage of coordinates that differ
`'jaccard'`	One minus the Jaccard coefficient, the percentage of nonzero coordinates that differ

Mathematical Definitions of Methods

Given an *m*-by-*n* data matrix `X`, which is treated as *m* (1-*by*-*n*) row vectors x_1, x_2, ..., x_m, the various distances between the vector x_r and x_s are defined as follows:

- Euclidean distance
- $d^2_{rs} = (x_r - x_s)(x_r - x_s)'$
- Standardized Euclidean distance
- $d^2_{rs} = (x_r - x_s)D^{-1}(x_r - x_s)'$
- where *D* is the diagonal matrix with diagonal elements given by v^2_j , which denotes the variance of the variable X_j over the *m* objects.
- Mahalanobis distance
- $d^2_{rs} = (x_r - x_s)V^{-1}(x_r - x_s)'$
- where *V* is the sample covariance matrix.
- City Block metric
- $d_{rs} = \sum_{j=1}^{n} |x_{rj} - x_{sj}|$
- Minkowski metric

- $$d_{rs} = \left\{ \sum_{j=1}^{n} |x_{rj} - x_{sj}|^p \right\}^{\frac{1}{p}}$$
- Notice that for the special case of *p* = 1, the Minkowski metric gives the City Block metric, and for the special case of *p* = 2, the Minkowski metric gives the Euclidean distance.
- Cosine distance
- $$d_{rs} = \left(1 - x_r x'_s / (x'_r x_r)^{\frac{1}{2}} (x'_s x_s)^{\frac{1}{2}}\right)$$
- Correlation distance
- $$d_{rs} = 1 - \frac{(x_r - \bar{x}_r)(x_s - \bar{x}_s)'}{[(x_r - \bar{x}_r)(x_r - \bar{x}_r)']^{\frac{1}{2}} [(x_s - \bar{x}_s)(x_s - \bar{x}_s)']^{\frac{1}{2}}}$$

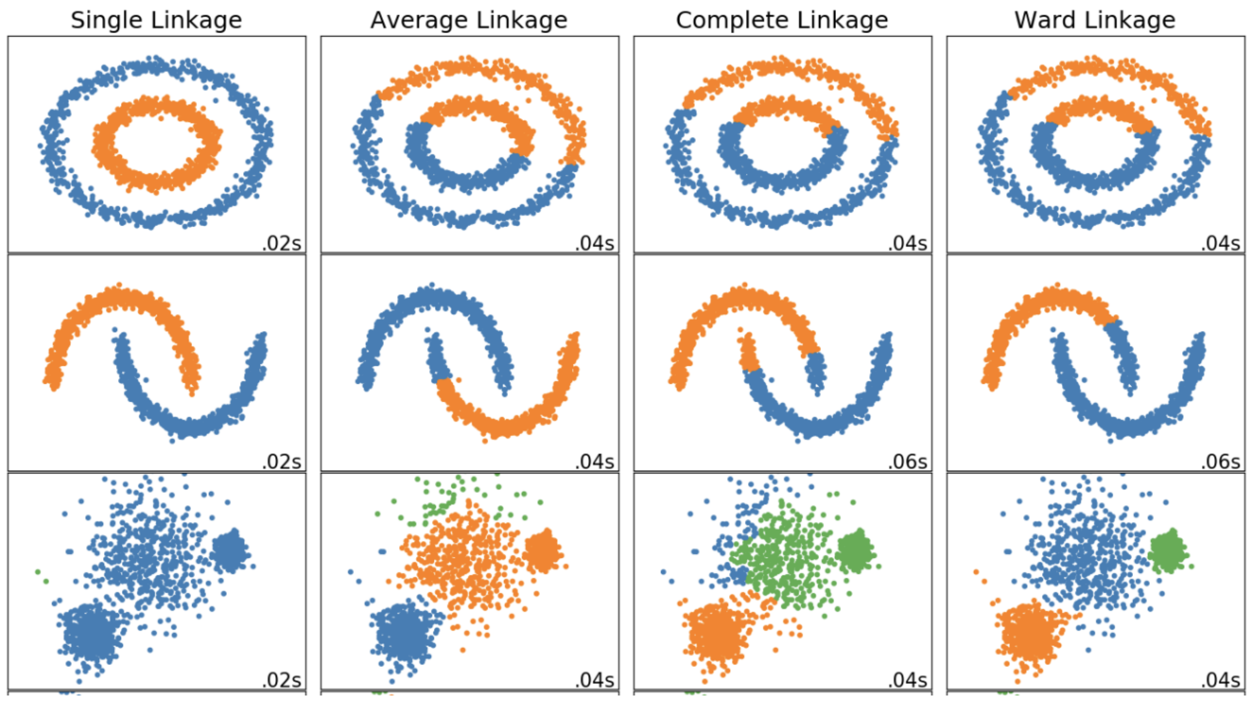
where

- $\bar{x}_r = \frac{1}{n} \sum_j x_{rj}$ and

  $\bar{x}_s = \frac{1}{n} \sum_j x_{sj}$
- Hamming distance
- $$d_{rs} = (\#(x_{rj} \neq x_{sj})/n)$$
- Jaccard distance
- $$d_{rs} = \frac{\#[(x_{rj} \neq x_{sj}) \wedge ((x_{rj} \neq 0) \vee (x_{sj} \neq 0))]}{\#[(x_{rj} \neq 0) \vee (x_{sj} \neq 0)]}$$

Linkage

Similar to gradient descent, you can tweak certain parameters to get drastically different results.

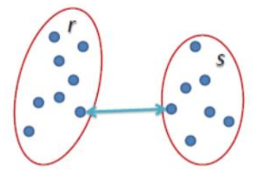


Description

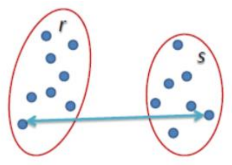
`Z = linkage(Y)` creates a hierarchical cluster tree, using the Single Linkage algorithm. The input matrix, `Y`, is a distance vector of length $((m-1)\cdot m/2)$-by-1, where *m* is the number of objects in the original dataset. You can generate such a vector with the `pdist` function. `Y` can also be a more general dissimilarity matrix conforming to the output format of `pdist`.

`Z = linkage(Y,'method')` computes a hierarchical cluster tree using the algorithm specified by `'method'`, where `'method'` can be any of the following character strings that identify ways to create the cluster hierarchy. Their definitions are explained in Mathematical Definitions.

`'single'`	Shortest distance (default)



$$L(r,s)=\min(D(x_{ri},x_{sj}))$$

`'complete'`	The distance between two clusters is the longest distance between two points in each cluster  $L(r,s)=\max(D(x_{ri},x_{sj}))$
`'average'`	Average distance
`'centroid'`	Centroid distance. The output `Z` is meaningful only if `Y` contains Euclidean distances.
`'ward'`	Incremental sum of squares

The output, `Z`, is an (*m*-1)-by-3 matrix containing cluster tree information. The leaf nodes in the cluster hierarchy are the objects in the original dataset, numbered from 1 to *m*. They are the singleton clusters from which all higher clusters are built. Each newly formed cluster, corresponding to row *i* in `Z`, is assigned the index *m*+*i*, where *m* is the total number of initial leaves.

Columns 1 and 2, `Z(i,1:2)`, contain the indices of the objects that were linked in pairs to form a new cluster. This new cluster is assigned the index value *m*+*i*. There are *m*-1 higher clusters that correspond to the interior nodes of the hierarchical cluster tree.

Column 3, `Z(i,3)`, contains the corresponding linkage distances between the objects paired in the clusters at each row *i*.

For example, consider a case with 30 initial nodes. If the tenth cluster formed by the `linkage` function combines object 5 and object 7 and their distance is 1.5, then row 10 of `Z` will contain the values (`5`, `7`, `1.5`). This newly formed cluster will have the index 10+30=40. If cluster 40 shows up in a later row, that means this newly formed cluster is being combined again into some bigger cluster.

dendrogram

Plot dendrogram graphs

Description

`H = dendrogram(Z)` generates a dendrogram plot of the hierarchical, binary cluster tree, `Z`. `Z` is an (`m`-1)-by-3 matrix, generated by the `linkage` function, where `m` is the number of objects in the original dataset.

A dendrogram consists of many U-shaped lines connecting objects in a hierarchical tree. Except for the Ward linkage (see `linkage`), the height of each U represents the distance between the two objects being connected. The output, `H`, is a vector of line handles.

`H = dendrogram(Z,p)` generates a dendrogram with only the top `p` nodes. By default, `dendrogram` uses 30 as the value of `p`. When there are more than 30 initial nodes, a dendrogram may look crowded. To display every node, set `p = 0`.

`[H,T] = dendrogram(...)` generates a dendrogram and returns `T`, a vector of length `m` that contains the leaf node number for each object in the original dataset. `T` is useful when `p` is less than the total number of objects, so some leaf nodes in the display correspond to multiple objects. For example, to find out which objects are contained in leaf node `k` of the dendrogram, use `find(T==k)`. When there are fewer than `p` objects in the original data, all objects are displayed in the dendrogram. In this case, `T` is the identity map, i.e., `T = (1:m)'`, where each node contains only itself.

When there are fewer than `p` objects in the original data, all objects are displayed in the dendrogram. In this case, `T` is the identity map, i.e., `T = (1:m)'`, where each node contains only itself.

`[H,T,perm] = dendrogram(...)` generates a dendrogram and returns the permutation vector of the node labels of the leaves of the dendrogram. `perm` is ordered from left to right on a horizontal dendrogram and bottom to top for a vertical dendrogram.

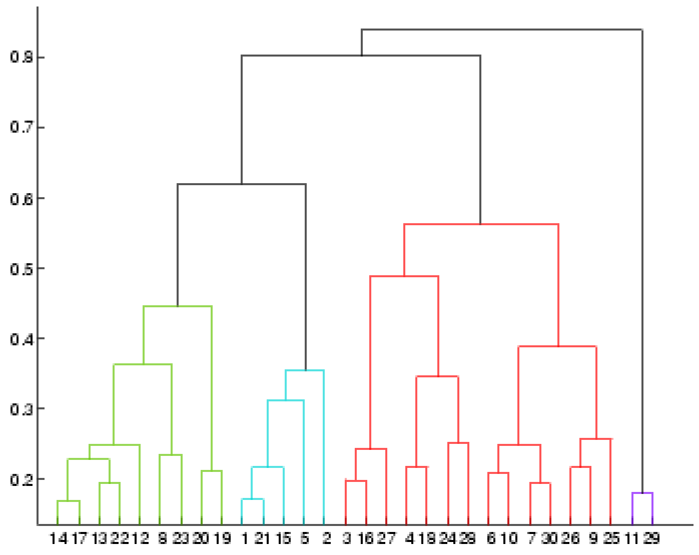
`[...] = dendrogram(...,'colorthreshold',t)` assigns a unique color to each group of nodes in the dendrogram where the linkage is less than the threshold `t`. `t` is a value in the interval `[0,max(Z(:,3))]`. Setting `t` to the string `'default'` is the same as t = .7(`max(Z(:,3)))`. `0` is the same as not specifying `'colorthreshold'`. The value `max(Z(:,3))` treats the entire tree as one group and colors it all one color.

`[...] = dendrogram(...,'orientation','`*`orient`*`')` orients the dendrogram within the figure window. The options for '*`orient`*' are

`'top'`	Top to bottom (default)
`'bottom'`	Bottom to top
`'left'`	Left to right
`'right'`	Right to left

Example

```
rand('seed',12);
X= rand(100,2);
Y= pdist(X,'cityblock');
Z= linkage(Y,'average');
[H,T] = dendrogram(Z,'colorthreshold','default');
```



```
find(T==20)
ans =
    20
    49
    62
    65
    73
    96
```

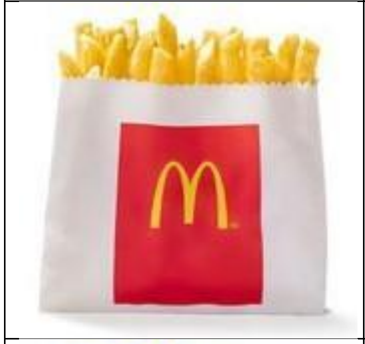
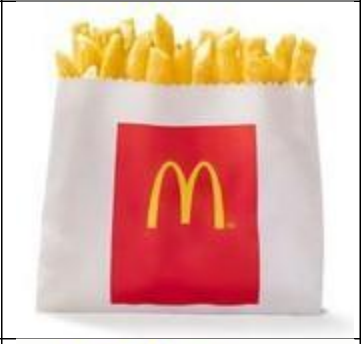
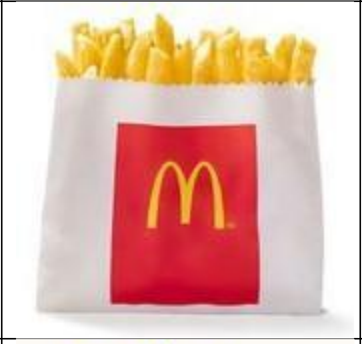
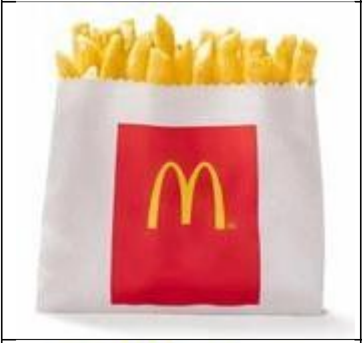
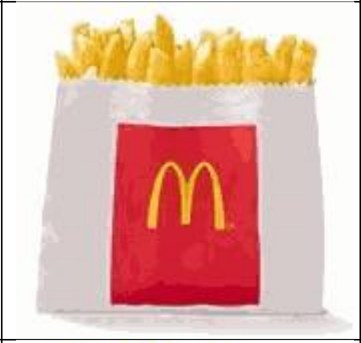
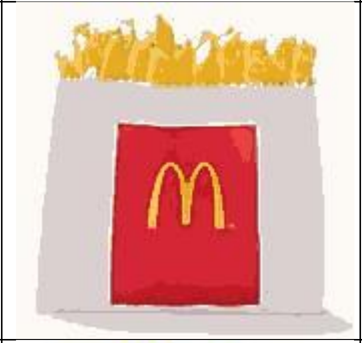
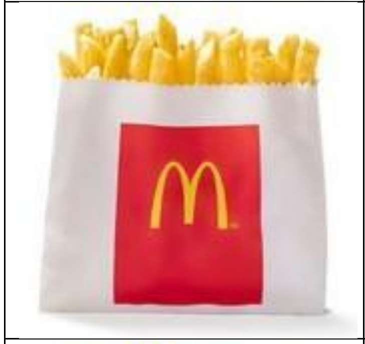
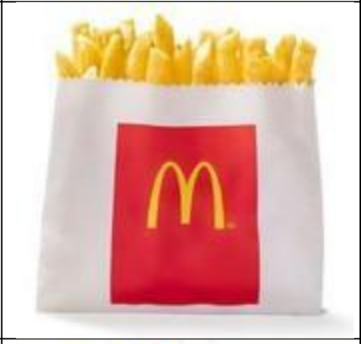
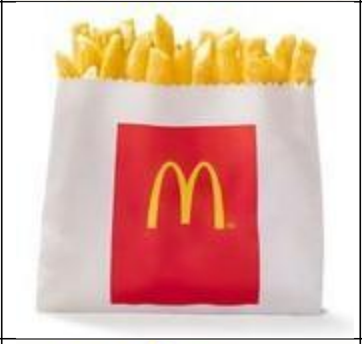
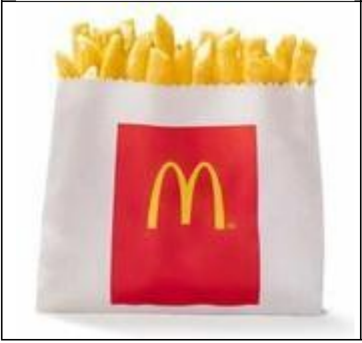
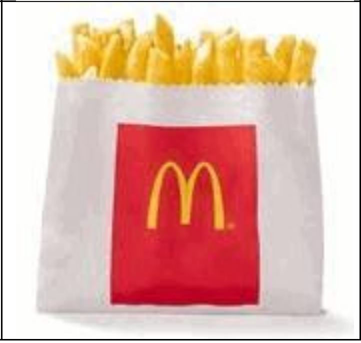
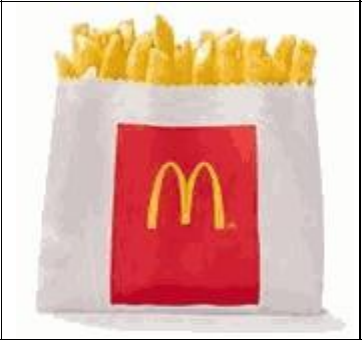
Code

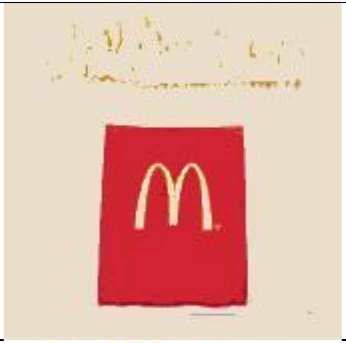
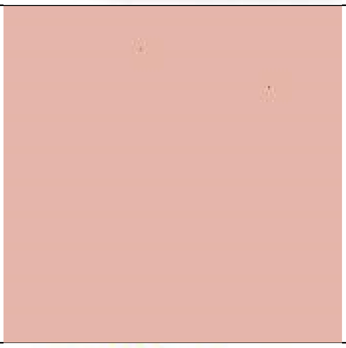
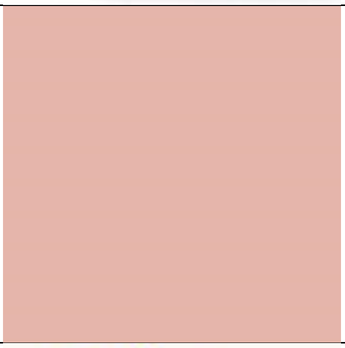
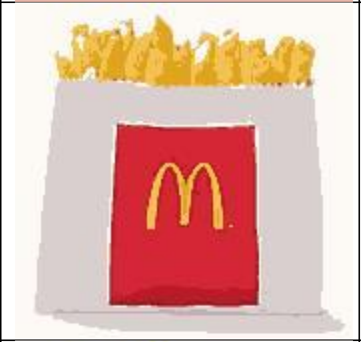
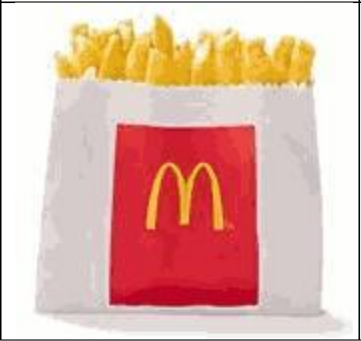
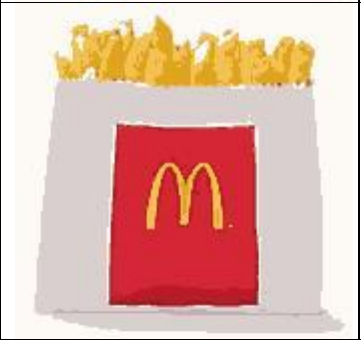
Linkage with complete and single and average and centroid methods:

```
def all_linkage(y):
    try:
        single_z=linkage(y,'single')
    except ValueError:
        print("ha")
    complete_z=linkage(y,'complete')
    average_z = linkage(y,'average')
    centroid_z=linkage(y,'centroid')
    return
{'single':single_z,'complete':complete_z,'average':average_z,'centroid':centroid_z}
pdist with Euclidean and seuclidean methods:
def all_pdist(x):
    euclidean_y=pdist(x,'euclidean')
    seuclidean_y = pdist(x, 'seuclidean')
    array_sum = np.sum(seuclidean_y)
    array_has_nan = np.isnan(array_sum)
    if array_has_nan:
        seuclidean_y=euclidean_y
    return {'euclidean':euclidean_y,'seuclidean':seuclidean_y}
clustering using linkage :
def all_dendogram(x,img):
    dict={}
    t=0.5
    # for i in range(10000, len(x), 10000):
    for key,value in all_pdist(x).items():
        for key1,value1 in all_linkage(value).items():
                try:

dict["linkage_method_"+key1+"_pdist_type_"+key+"_t_"+str(t)].extend(fcluster(value1,c
riterion='distance',t=t))
                except KeyError:
                    dict["linkage_method_" + key1 + "_pdist_type_" + key + "_t_" +
str(t)]=[]
                    dict["linkage_method_" + key1 + "_pdist_type_" + key + "_t_" +
str(t)].extend(fcluster(value1, criterion='distance',t=t))
    # if i<len(x):
    #     for key,value in all_pdist(x[i:]).items():
    #         for key1,value1 in all_linkage(value).items():
    #
dict["linkage_method_"+key1+"pdist_type_"+key+"_t_0.1"].extend(fcluster(value1,criter
ion='distance',t=0.1))

    for key,array in dict.items():
        identifiers=get_label_dict(x,array)
         imag_filtering(array,img,identifiers,key)
image filtering:
def imag_filtering(y, img: Image, identifiers,save_name):
    new_image = []
    for x in range(img.size[1]):
        new_image_row = []
        for i in range(img.size[0]):
            new_image_row.append(identifiers[y[x * (img.size[0]) + i]])
        new_image.append(new_image_row)
    new_image = np.asarray(new_image, dtype=np.uint8)
    new_image = Image.fromarray(new_image, 'RGB')
    new_image.save(save_name+'.jpg')
    new_image.show()
results:
```

t=0.1	t=0.5	t=1	Linkage and pdist methods
			Linkage: Average Pdist: euclidean
			Linkage: Average Pdist: seuclidean
			Linkage: complete Pdist: euclidean
			Linkage: complete Pdist: seuclidean

			Linkage: single Pdist: euclidean
			Linkage: single Pdist: seuclidea n
			Linkage: Centroid Pdist: euclidean
			Linkage: Centroid Pdist: seuclidea n

References:

https://python-graph-gallery.com/dendrogram/
https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn/
https://scikit-learn.org/stable/auto_examples/cluster/plot_agglomerative_dendrogram.html