CLUSTERING IN MACHINE LEARNING

Background

Clustering is in simple terms grouping most (silimilar items together and most different items in separate groups. This similarity we use on daily basis in our nearby environment for grouping objects like good apples, food or even good friends. We generally tend to choose some set of features like color,shape,texture,taste to calculate the likeliness/similarity between them.It need not be just one feature which sometimes help us in grouping objects. It could be multivariate similarity. Clustering in machine learning is synonymous emulation process for machines to search for similar objects.

Objective

The goal of this notebook is walk through clustering in context of machine learning and give thorough understanding about some one main type of clustering often used i.e. Kmeans method. We will then see with implementation of kmeans and kmeans++ algorithm how clustering is used for classfying similar groups and identifying its lables with some examples ranging from tabular data to images.Later we will touch upon possible challenges in our algorithm and future scope of improvements.

For curious souls, there is a last section for further reads !!!

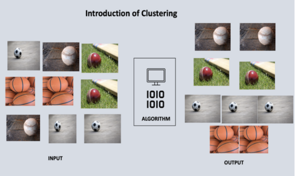
Index

Introduction to Clustering

Why Clustering is Unsupervised Learning?

Clustering falls under unsupervised machine learning as machine unlike us sometimes maynot have clear labels attached to some of the objects to start with. E.g by looking at a apple we know that it's an apple because our mind is trained to see a lot of objects like apple since childhood. But let's say a company is running curated sets of marketing ads and want to segment users suitable for individual ads. To calculate similarity between users, it would require some information like user profile, background purchase history and other information since they don't have exact prior external label to group users. Hence, it is a learning which is unsupervised and learnt based on likeliness scores. So, these features becomes these factors for learning and to calculate similarity index.

Out[108]:



Above image demonstrates how machine finds a pattern to segregate different type of balls.

- To start with it has all different types of balls together and every ball has some features (in form of binary coded values here).
- Machine which is our clustering algorithm used will try to identify patterns in these computer coded features.
- And voila, it's able to find three different sets of balls (basketball, football and volleyball)

Now next question would arise how many curated ads should they run?

Now the next obvious question is how many groups should you choose so that all your data will be grouped together. This is subjective for humans but for machines it is one of the most important parameter to find out ideal number of clusters. Some algorithms requires to mention clusters for which there are ways to find optimal e.g. Elbow Method or Silhoutte cofficient. Some maynot need them like hierarchical method of clustering(explained a little later).

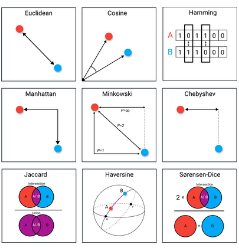
Similarity of Cluster

There are different distance measures for calculating similarity between two values ranging from simple euclidean distan, manhattan to cosine similarity. Choice is dependent on the type(categorical or numerical) and dimensionality of data we are dealing with. In my implemenation of kmeans and kmeans++ , I will be using Euclidean distance for simplicity.

Out[109]:

Click here to toggle on/off the raw code.

Out[18]:



Credits: https://towardsdatascience.com/9-distance-measures-in-data-science-918109d069fa

Types of Clustering

There are different methods of clustering depending on difference in base algorithm used. Most common broad types are: partition and hierarchical clustering.

- Partition(centroid based) again can be done from differet ways,e.g. kmeans/k medians algorithm,mean shift and spatial clustering algorithm (density based), expectation maximising algorithm based on gaussian mixture models(clusters maynot be circular).
- Hierarchical may include top-down(divisive) or bottom-up (hierarchical agglomerative clustering), these approaches have one cluster to branching to sub clusters ot all points in separate clusters to clubbing them in broader groups. There are further model based clustering like RF.

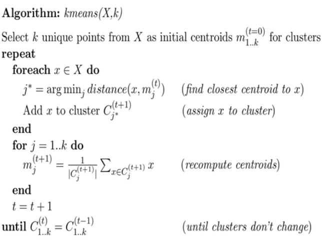
Kmeans Algorithm

It is an iterative process in which it tries to minimize the distance of the data points from the centroid points. It’s used to group the data points into k number of clusters based on their similarity. Euclidean distance is used to calculate the similarity.

Advantages : It is easy to implement and faster than most of the clustering algorithms. It also provides clear tight boundaries unlike hierarchical clustering.

Disadvantages: Kmeans as take random points initially as centroids, so it can give different results on each run. hence, it's less stable than kmeans++. Also it performs poorly on nested joint data points(will see later).

Out[115]:

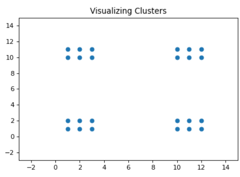


```
Algorithm: $kmeans(X,k)$
Select $k$ unique points from $X$ as initial centroids $m_{1..k}^{(t=0)}$ for clusters $C_{1..k}^{(t=0)}$
repeat
    foreach $x \in X$ do
        $j^* = \arg\min_j distance(x, m_j^{(t)})$        $(find\ closest\ centroid\ to\ x)$
        Add $x$ to cluster $C_{j^*}^{(t+1)}$        $(assign\ x\ to\ cluster)$
    end
    for $j = 1..k$ do
        $m_j^{(t+1)} = \frac{1}{|C_j^{(t+1)}|} \sum_{x \in C_j^{(t+1)}} x$        $(recompute\ centroids)$
    end
    $t = t + 1$
until $C_{1..k}^{(t)} = C_{1..k}^{(t-1)}$        $(until\ clusters\ don't\ change)$
```

Let us understand this with one example of points shown below.

- **Step 1** - We can assume to take take 2 clusters (k) initially .
- **Step 2** - Two points are randomaly initlaised to be taken for icentroid initialisation. Let's say we pick, (2,2) and (2,10)
- **Step 3** - Calculate distance of all points in data from these two centroids using euclidean distance and assign each point to closest centroid (its cluster).Here, cluster (2,2) will have all below graph points and above in (2,10)
- **Step 4** - Update the centroid value by calculating mean of all points falling in the cluster. Now, the centroid will move to white spaces between above groups and below groups of graph(between 4-8).
- **Step 5** - Repeat above process of reassignment of data points to new clusters.Iterate Steps (3 & 4) until centroid position keeps changing(high tolerance) or maximum number of iterations is achieved.

We can repeat above processing by changing number of cluster k= 4(can use elbow method to get) here and see if our clusters are more distinguishable.



Implementation of code is divided in clear blocks of tasks at hand:

1. Random initialising centroids using python random choice
2. Distance Metric function , here euclidean distance
3. Kmeans main loop for iterating process of reassigning centroid based on mean of cluster.

Centroid Initialisation for kmeans

```
random_index = np.random.choice(len(X), k, replace=False)
ctds = X[random_index, :]
ctds= np.array(ctds)
```

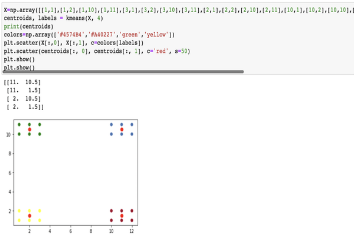
Iterative reassignment of centroids

```
for i in range(max_iter):

    distances=distance_sqrt(X,ctds)
    closest_ctd = np.argmin(distances, axis=1)

    K, D = ctds.shape
    new_ctds = np.empty(ctds.shape)
    for j in range(ctds.shape[0]):

        new_ctds[j] = np.mean(X[closest_ctd == j], axis = 0)

        if np.sum((new_ctds - ctds)/ctds * 100.0) > tolerance:
            break

    ctds=new_ctds

    distances=distance_sqrt(X,ctds)
    closest_ctd = np.argmin(distances, axis=1)
```

Examples of Kmeans

The result of kmeans on previous data: when clustered with k=2 and when clustered with k=4. We can see ideal clusters could be k=4 here.

Out[38]:



```
X=np.array([[1,1],[1,2],[1,10],[1,11],[3,1],[3,2],[3,10],[3,11],[2,1],[2,2],[2,10],[2,11],[10,1],[10,2],[10,10],[
centroids, labels = kmeans(X, 4)
print(centroids)
colors=np.array(['#4574B4','#A40227','green','yellow'])
plt.scatter(X[:,0], X[:,1], c=colors[labels])
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=50)
plt.show()
plt.show()
```

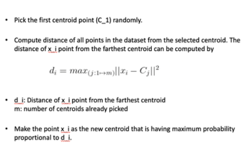
```
[[11.  10.5]
 [11.   1.5]
 [ 2.  10.5]
 [ 2.   1.5]]
```

Kmeans++ Algorithm

Kmeans++ is different from kmeans in only process of finding first set of centroids which is done randomly in kmeans. But in kmeans++ , we try to find k centroids across data points such that all each are at maximum distance from other clusters.

Advantage of kmeans++ over kmeans is that it convergest faster because of final centroids may lie somewhere near to initial one. Also it shows genrally better performance in image compression(which we can see below).

Out[111]:



- Pick the first centroid point (C_1) randomly.
- Compute distance of all points in the dataset from the selected centroid. The distance of x_i point from the farthest centroid can be computed by

$$d_i = max_{(j:1 \mapsto m)} ||x_i - C_j||^2$$

- d_i: Distance of x_i point from the farthest centroid
  m: number of centroids already picked
- Make the point x_i as the new centroid that is having maximum probability proportional to d_i.

- Repeat the above two steps till you find k-centroids.

Implementation of Kmeans++:

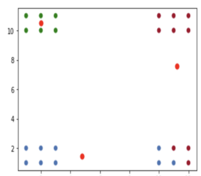
```
def kmeans_plus_plus(X, k):
    ctds = []
    X = np.array(X)

    first_idx = np.random.choice(range(X.shape[0]), )
    ctds.append(X[first_idx, :].tolist())

    for i in range(k - 1):
        distance = dist(X, np.array(ctds))
        min_dist = np.min(distance, axis = 1)
        max_idx = np.argmax(min_dist, axis = 0)
        new_ctds = X[max_idx, :]
        ctds.append(new_ctds.tolist())

    return np.array(ctds)
```

Kmeans++ Example

For example, if we pick k =3 in previous dataset, kmeans will always pick a triangle in first initialization to maximize distances between all three centroids. Similarly, kmeans++ will always give 4 clear clusters in its first initialization only for this data to maximuze distance between centroids.

Out[47]:

```
[[ 4.77777778  1.44444444]
 [11.22222222  7.55555556]
 [ 2.         10.5       ]]
```

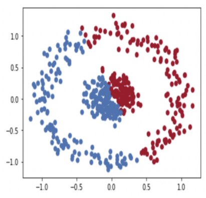


Drawbacks of Kmeans and Kmeans++

We can see poor performance of kmeans or kmeans++ of disjoint and nested structures where it's not able to pick internal cluster as separate.These can be avoided with other methods of clustering like spectral clustering

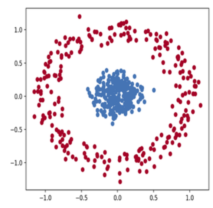
Out[120]:

```
[[-0.3285314  -0.13076984]
 [ 0.40353611  0.18325268]]
```



We can see below spectral clustering gives clearly seperated clusters in this cases.

Out[49]:

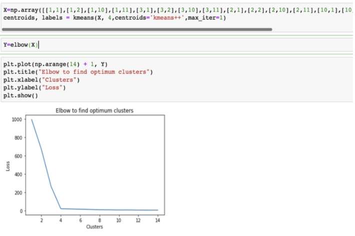


Elbow Method to Find optimum Clusters

The elbow method helps in finding the optimal number of clusters for k-means clustering. The elbow method plots the value of the loss for different values of k. If k increases, average loss will decrease since each cluster will have fewer points, and tthere will be less loss of data. But we want to find out that k when the decrease in loss becomes almost flat line i.e. we need to find k for elbow in plot.

Diasdv: This process requires running clustering for all possible k ranges gives to it. Hence, for large datasets can be slow.

Out[138]:



```
X=np.array([[1,1],[1,2],[1,10],[1,11],[3,1],[3,2],[3,10],[3,11],[2,1],[2,2],[2,10],[2,11],[10,1],[10
centroids, labels = kmeans(X, 4,centroids='kmeans++',max_iter=1)
```

```
Y=elbow(X)
```

```
plt.plot(np.arange(14) + 1, Y)
plt.title("Elbow to find optimum clusters")
plt.xlabel("Clusters")
plt.ylabel("Loss")
plt.show()
```

Image Compression Examples based on Kmeans++ Clustering

One of the common application of clustering is infield of Image compression.KMeans Clustering algorithm takes advantage of the visual perception of the human eyes and it will use few colors to represent the same image. Colors having different values of intensity that are RGB values seem the same to the human eye. The K-Means algorithm takes this advantage and clubs similar looking colors(pixel values). This may hamper image quality but gives ability to store big images as small.

Let us understand it with one example of grayscale image.Our kmeans++ algorithm is flexible enough to be used in it. It just needs image as array of pixel values to label them. For this:

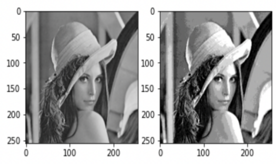
- **Step 1** - Load image in python using imread as array of pixel values .
- **Step 2** - The size of the input image could be (rows, cols, 3) or (rows, cols) (in grayscale), flatten all the pixel values to a single dimension of size (rows*cols) and its elements will either 3 RGB values for colored image or rone pixel value for grayscale.
- **Step 3** - Implement k-Means++ algo to find k-centroid points which will be its surrounding color approximation pixel values.
- **Step 4** - Replace the value of each of the pixels with its ccentroid of its cluster and reshape the image to its actual dimension of (rows,cols,3) or (rows,cols).

GrayScale Image Example

One of the grayscale images is compressed to 8 clusters(different pixel values). We can see the prformance is good for 8 clusters.Though lossy compression is visible to a human eye, actual content is not lost at cost of reduction of size.

```
image = io.imread('/Users/surbhiprasad/dsa/lena.png')
io.imshow(image)
io.show()

rows = image.shape[0]
cols = image.shape[1]
zeros=np.zeros(len(np.array(image).reshape(-1,1)))
combined = np.hstack((np.array(image).reshape(-1,1), zeros.reshape(-1,1)))
X=image

k=8
centroids, labels = kmeans(combined, k=k,centroids='kmeans++', tolerance=.01)
centroids = centroids.astype(np.uint8)
X = centroids[labels] # reassign all points
final_image_pixels=np.array([x[0] for x in X])
```

Out[72]:

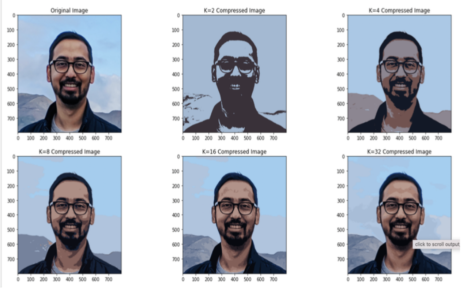


Colored Image Example

The process followed is similar , it's just that now we have three pixel values(RGB) per element of array. For colored images also, the algorithm perform well.

Image Credits: As my fiance was going crazy to include him in the project, I thought of this being the perfect way out.

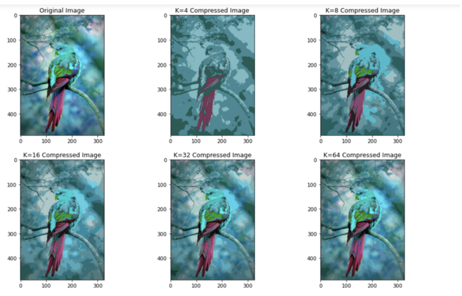
Out[96]:



From images, that k=2 and k=4 clusters aren't good to identigy with image.Also, it's visible that with k=8 clusters(just 8 RGB values), we have reduced image and there seems to have some significant lossof data as well(clouds and mountains) compared to first image(original).Hence, with clusters k=32 , we do get a decent compressed image picking facial uneven shaded, clouds, mountain colors as well.

Let us try on another example with many variations of color to see performance. We see below that the image after k=16 clusters are acceptable with not much distortion.

Out[129]:



Future Scope

- We can try different methods of distance calculation to check for imporvement in clusters, as we can see there is scope
- We can further explore other methods of clustering like spectral and model based algorithm to compare performance on similar data.
- Other initialization methods can be explored

Summary

We covered the goal of this project starting with what is clustering. Then we got to see why clustering is unsupervised machine learning and how similar it is to our real world scenarios. We further got a glipmse of how similarity is found by machines, how many clusters we should keep. We touched briefly upon types of clustering algorithms available of which we deep dived into Kmeans Clustering method. We implemented the same and saw its resulst on tabular and image data. Then we touched upon kmeans++ method of centroid initialization and compared two algorithms. We saw some shortcoming attached with theses algorithm on nested data. Then we explored further Image compression with Kmeans using grayscale and colored images. And we concluded with exploring elbow method to find the right cluster and reduce manual intervention.

Hence, there is scope for further explorations for improving current algorithm using different distance metrics and exploring other clustering algorithm for nested datasets.

Reading Material

A. Kmeans: KMeans Clustering

B. Image Compression: Image Compression using Kmeans

C. Types of Clustering: Types of Clustering

D. Different Distances: Distance Metrics

E. Applications of clustering Applications