```
result = svm.predict_all(testData)

#######   Check Accuracy   ########################
mask = result==responses
correct = np.count_nonzero(mask)
print correct*100.0/result.size
```

This particular technique gave me nearly 94% accuracy. You can try different values for various parameters of SVM to check if higher accuracy is possible. Or you can read technical papers on this area and try to implement them.

Additional Resources

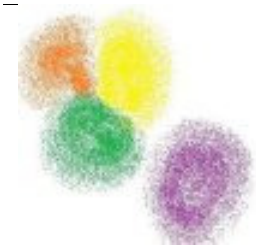
1. Histograms of Oriented Gradients Video

Exercises

1. OpenCV samples contain `digits.py` which applies a slight improvement of the above method to get improved result. It also contains the reference. Check it and understand it.

1.8.3 K-Means Clustering

- *Understanding K-Means Clustering*



Read to get an intuitive understanding of K-Means Clustering

- *K-Means Clustering in OpenCV*

Now let's try K-Means functions in OpenCV

Understanding K-Means Clustering

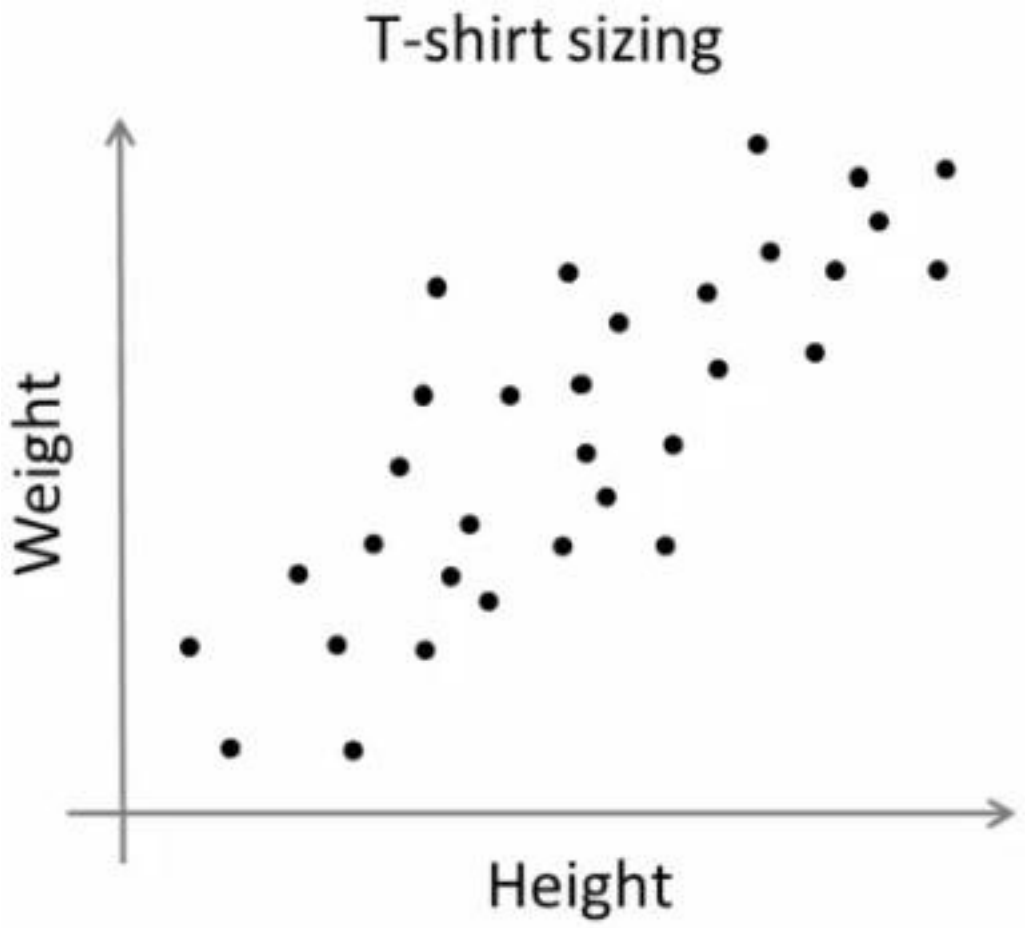
Goal

In this chapter, we will understand the concepts of K-Means Clustering, how it works etc.

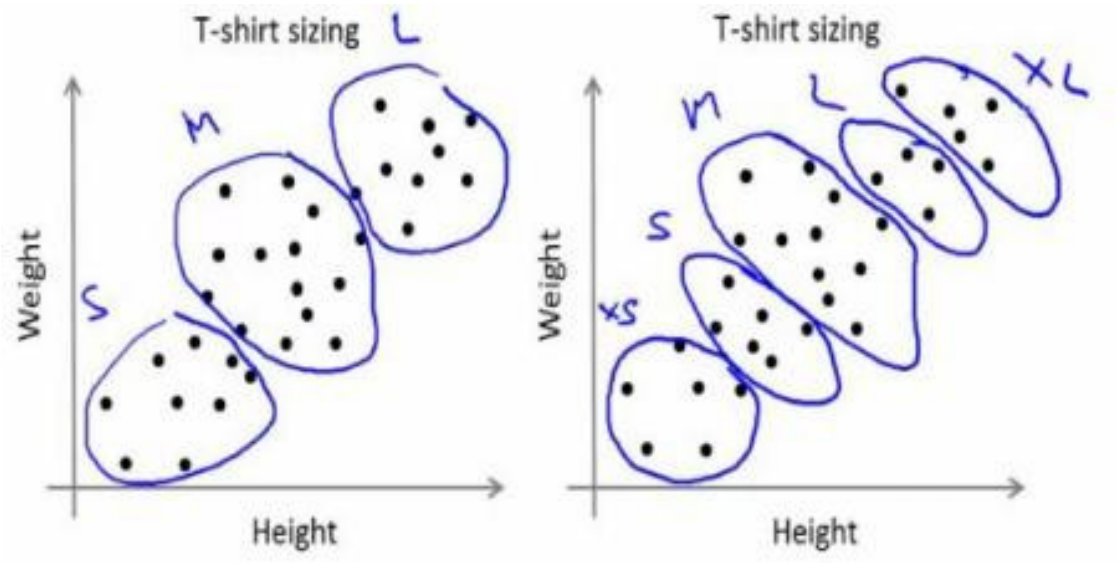
Theory

We will deal this with an example which is commonly used.

T-shirt size problem Consider a company, which is going to release a new model of T-shirt to market. Obviously they will have to manufacture models in different sizes to satisfy people of all sizes. So the company make a data of people's height and weight, and plot them on to a graph, as below:

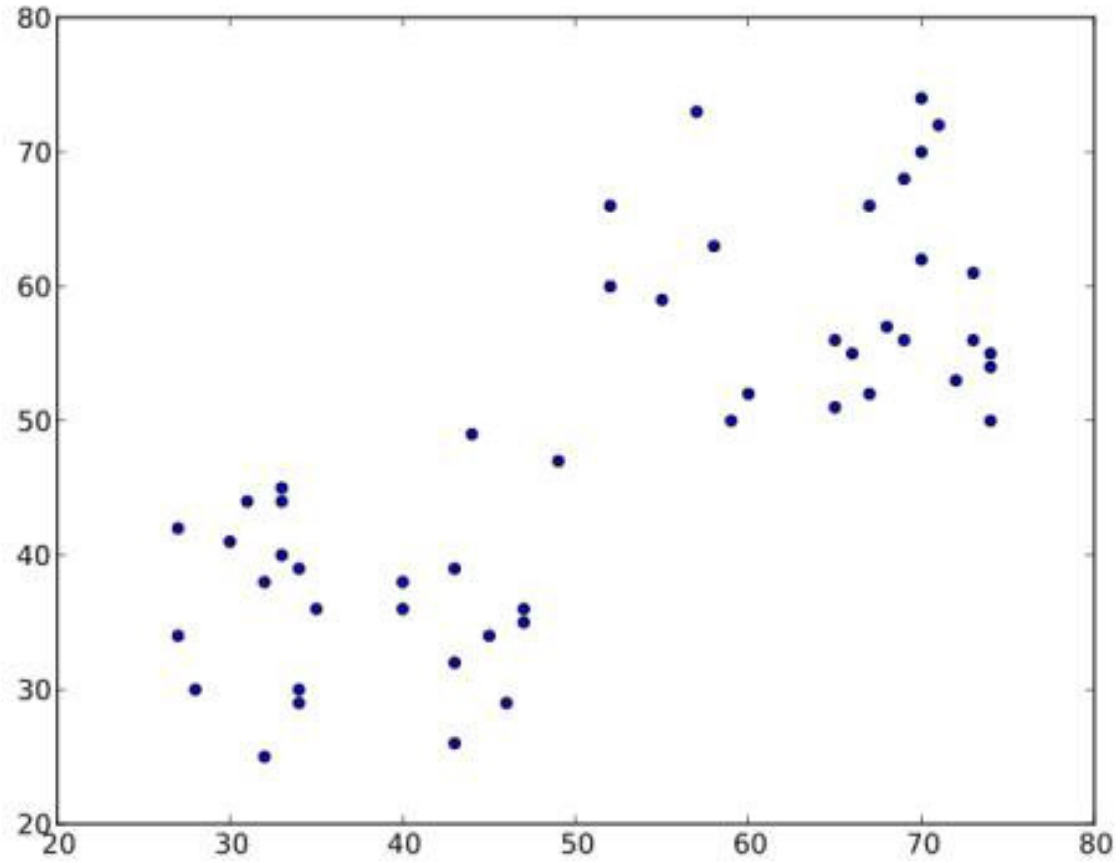


Company can't create t-shirts with all the sizes. Instead, they divide people to Small, Medium and Large, and manufacture only these 3 models which will fit into all the people. This grouping of people into three groups can be done by k-means clustering, and algorithm provides us best 3 sizes, which will satisfy all the people. And if it doesn't, company can divide people to more groups, may be five, and so on. Check image below :



How does it work ? This algorithm is an iterative process. We will explain it step-by-step with the help of images.

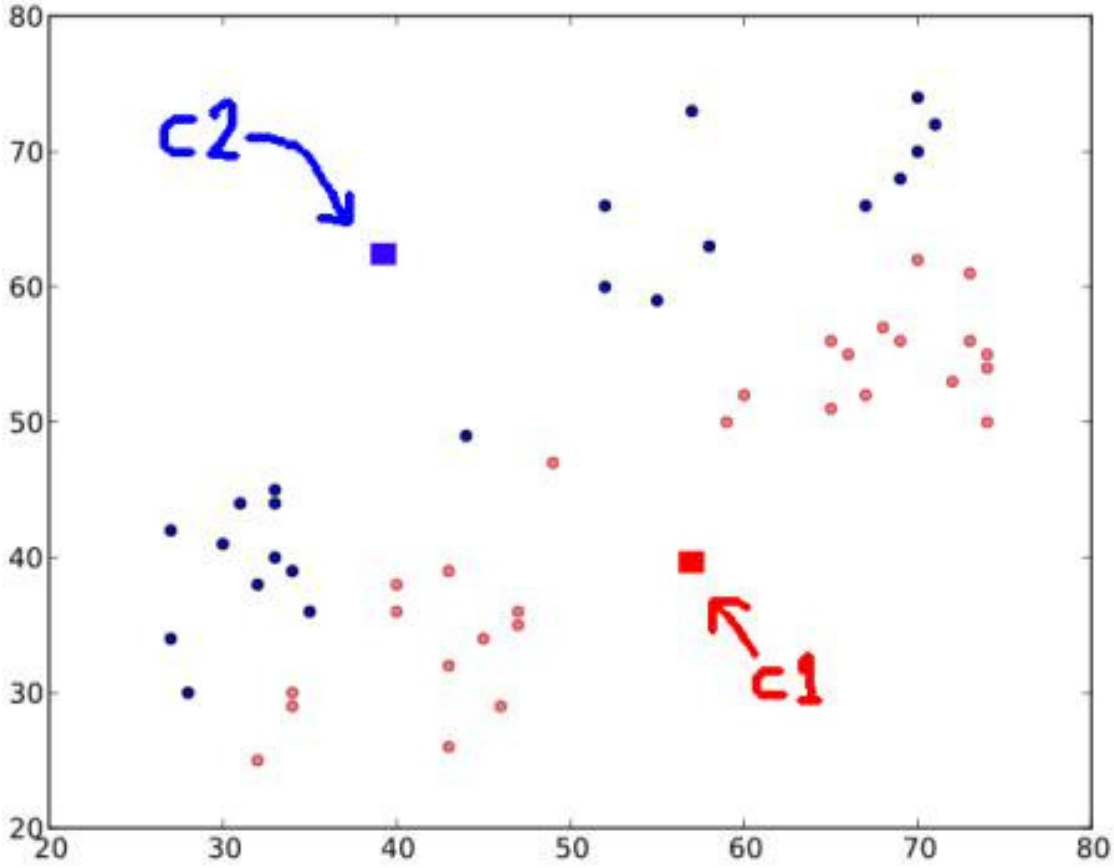
Consider a set of data as below (You can consider it as t-shirt problem). We need to cluster this data into two groups.



Step : 1 - Algorithm randomly chooses two centroids, $C1$ and $C2$ (sometimes, any two data are taken as the centroids).

Step : 2 - It calculates the distance from each point to both centroids. If a test data is more closer to $C1$, then that data is labelled with '0'. If it is closer to $C2$, then labelled as '1' (If more centroids are there, labelled as '2','3' etc).

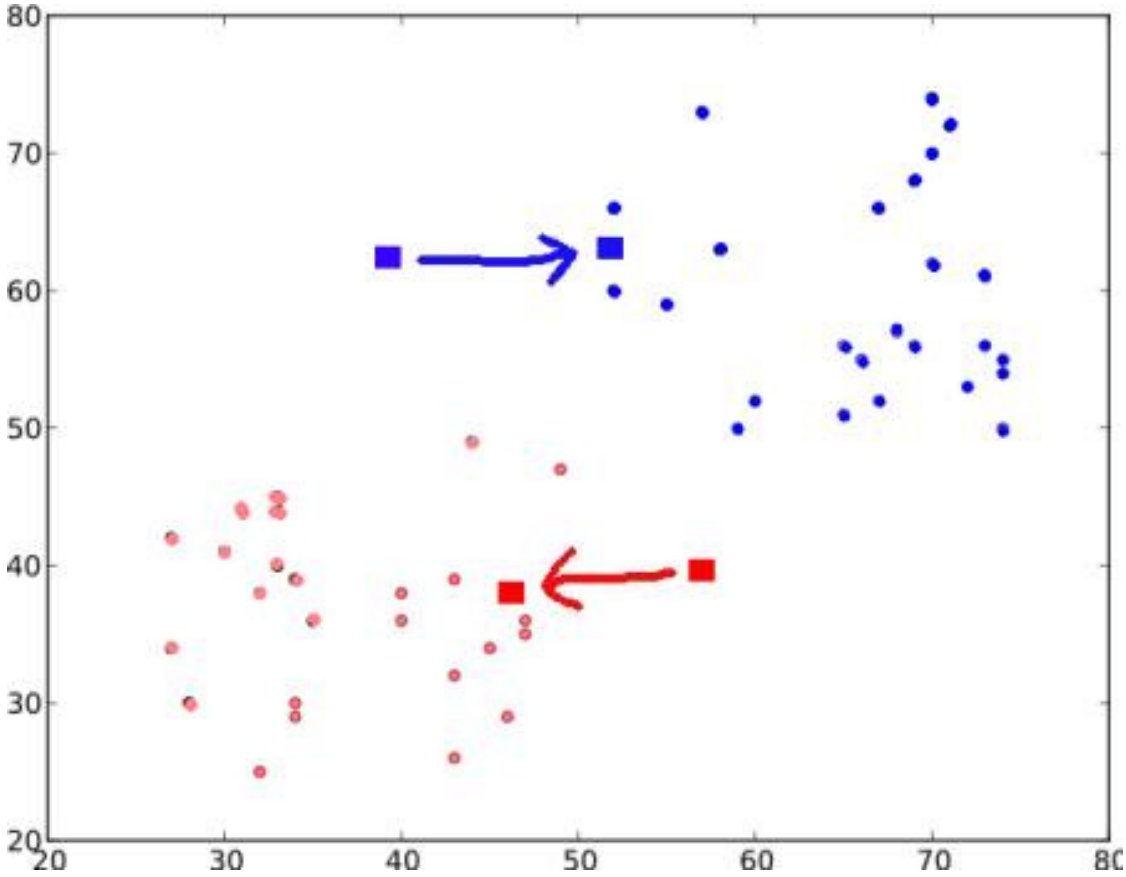
In our case, we will color all '0' labelled with red, and '1' labelled with blue. So we get following image after above operations.



Step : 3 - Next we calculate the average of all blue points and red points separately and that will be our new centroids. That is $C1$ and $C2$ shift to newly calculated centroids. (Remember, the images shown are not true values and not to true scale, it is just for demonstration only).

And again, perform step 2 with new centroids and label data to '0' and '1'.

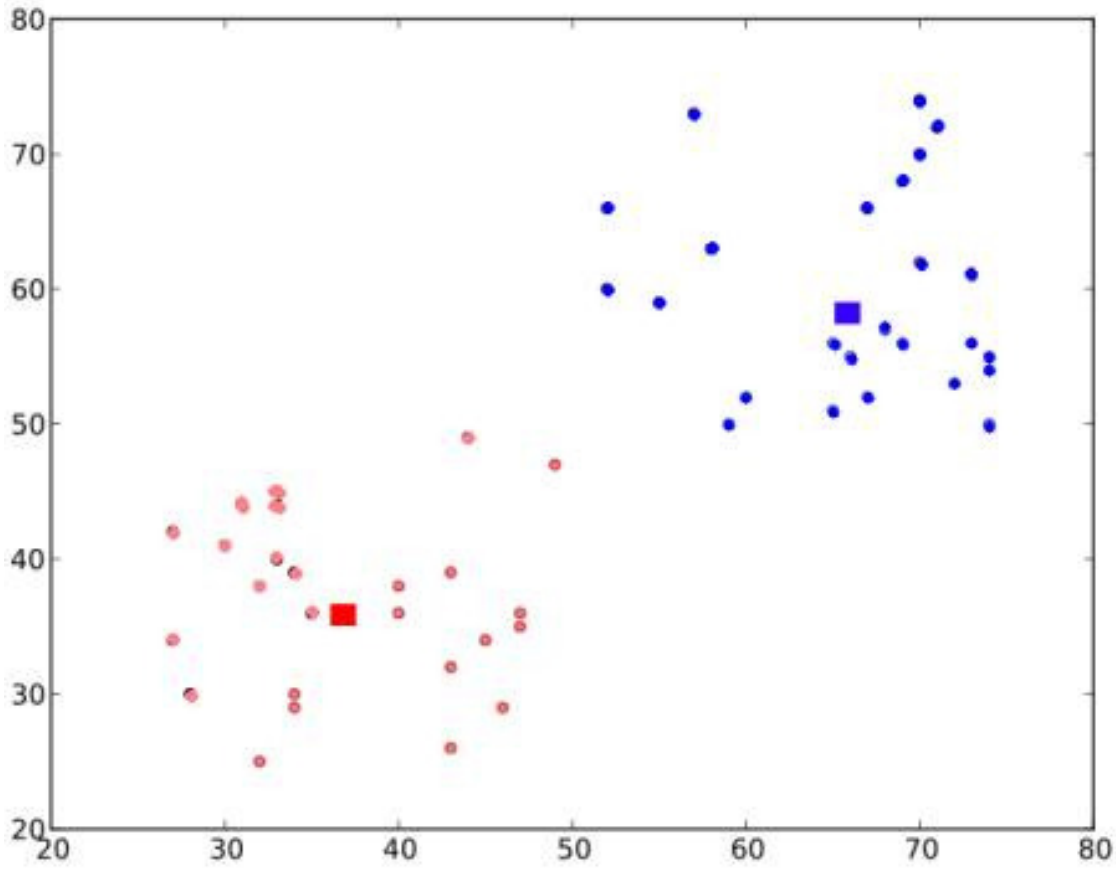
So we get result as below :



Now **Step - 2** and **Step - 3** are iterated until both centroids are converged to fixed points. *(Or it may be stopped depending on the criteria we provide, like maximum number of iterations, or a specific accuracy is reached etc.)* **These points are such that sum of distances between test data and their corresponding centroids are minimum**. Or simply, sum of distances between $C1 \leftrightarrow Red\_Points$ and $C2 \leftrightarrow Blue\_Points$ is minimum.

$$minimize \left[J = \sum_{All\ Red_Points} distance(C1, Red\_Point) + \sum_{All\ Blue\_Points} distance(C2, Blue\_Point) \right]$$

Final result almost looks like below :



So this is just an intuitive understanding of K-Means Clustering. For more details and mathematical explanation, please read any standard machine learning textbooks or check links in additional resources. It is just a top layer of K-Means clustering. There are a lot of modifications to this algorithm like, how to choose the initial centroids, how to speed up the iteration process etc.

Additional Resources

1. Machine Learning Course, Video lectures by Prof. Andrew Ng (Some of the images are taken from this)

Exercises

K-Means Clustering in OpenCV

Goal

- Learn to use **cv2.kmeans()** function in OpenCV for data clustering

Understanding Parameters

Input parameters

1. **samples** : It should be of **np.float32** data type, and each feature should be put in a single column.
2. **nclusters(K)** : Number of clusters required at end

3. **criteria** [It is the iteration termination criteria. When this criteria is satisfied, algorithm iteration stops. Actually, it should be a tuple of 3 parameters. They are `( type, max_iter, epsilon )`:]
    - **3.a - type of termination criteria** [It has 3 flags as below:] **cv2.TERM_CRITERIA_EPS** - stop the algorithm iteration if specified accuracy, *epsilon*, is reached. **cv2.TERM_CRITERIA_MAX_ITER** - stop the algorithm after the specified number of iterations, *max_iter*. **cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER** - stop the iteration when any of the above condition is met.
    - 3.b - max_iter - An integer specifying maximum number of iterations.
    - 3.c - epsilon - Required accuracy
4. **attempts** : Flag to specify the number of times the algorithm is executed using different initial labellings. The algorithm returns the labels that yield the best compactness. This compactness is returned as output.
5. **flags** : This flag is used to specify how initial centers are taken. Normally two flags are used for this : **cv2.KMEANS_PP_CENTERS** and **cv2.KMEANS_RANDOM_CENTERS**.

Output parameters

1. **compactness** : It is the sum of squared distance from each point to their corresponding centers.
2. **labels** : This is the label array (same as 'code' in previous article) where each element marked '0', '1'.....
3. **centers** : This is array of centers of clusters.

Now we will see how to apply K-Means algorithm with three examples.

1. Data with Only One Feature

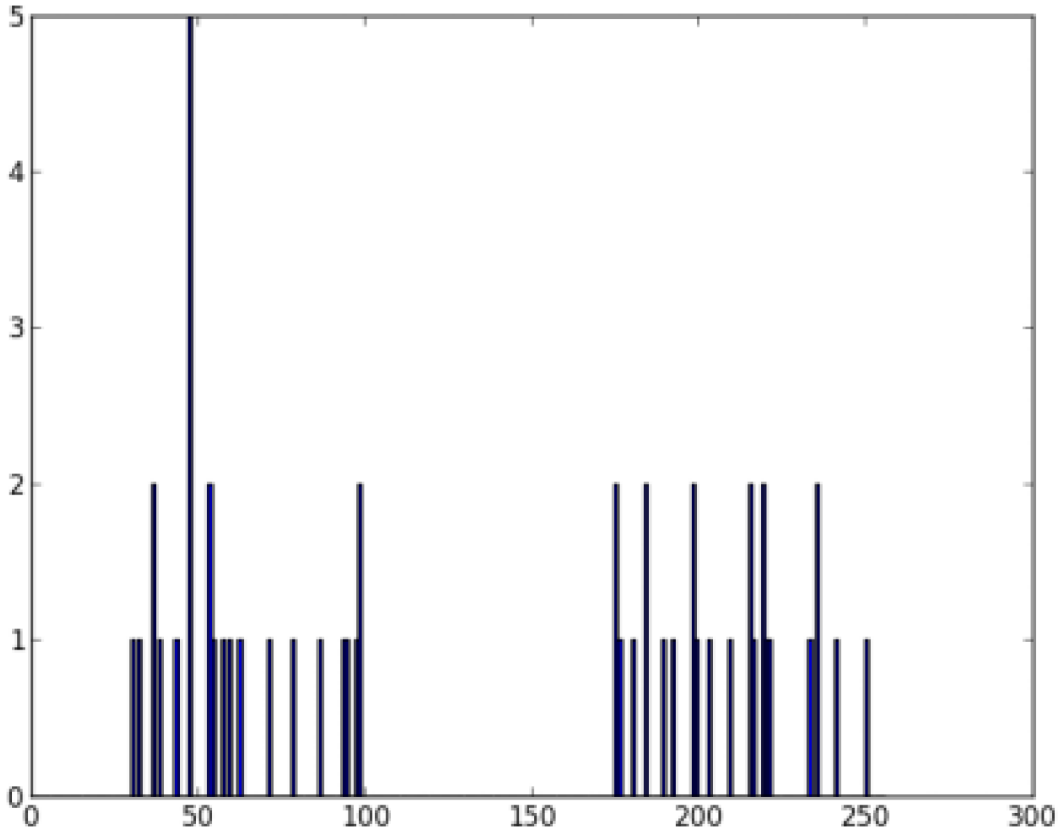
Consider, you have a set of data with only one feature, ie one-dimensional. For eg, we can take our t-shirt problem where you use only height of people to decide the size of t-shirt.

So we start by creating data and plot it in Matplotlib

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

x = np.random.randint(25,100,25)
y = np.random.randint(175,255,25)
z = np.hstack((x,y))
z = z.reshape((50,1))
z = np.float32(z)
plt.hist(z,256,[0,256]),plt.show()
```

So we have 'z' which is an array of size 50, and values ranging from 0 to 255. I have reshaped 'z' to a column vector. It will be more useful when more than one features are present. Then I made data of np.float32 type.

We get following image :



Now we apply the KMeans function. Before that we need to specify the *criteria*. My criteria is such that, whenever 10 iterations of algorithm is ran, or an accuracy of `epsilon = 1.0` is reached, stop the algorithm and return the answer.

```
# Define criteria = ( type, max_iter = 10 , epsilon = 1.0 )
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)

# Set flags (Just to avoid line break in the code)
flags = cv2.KMEANS_RANDOM_CENTERS

# Apply KMeans
compactness,labels,centers = cv2.kmeans(z,2,None,criteria,10,flags)
```

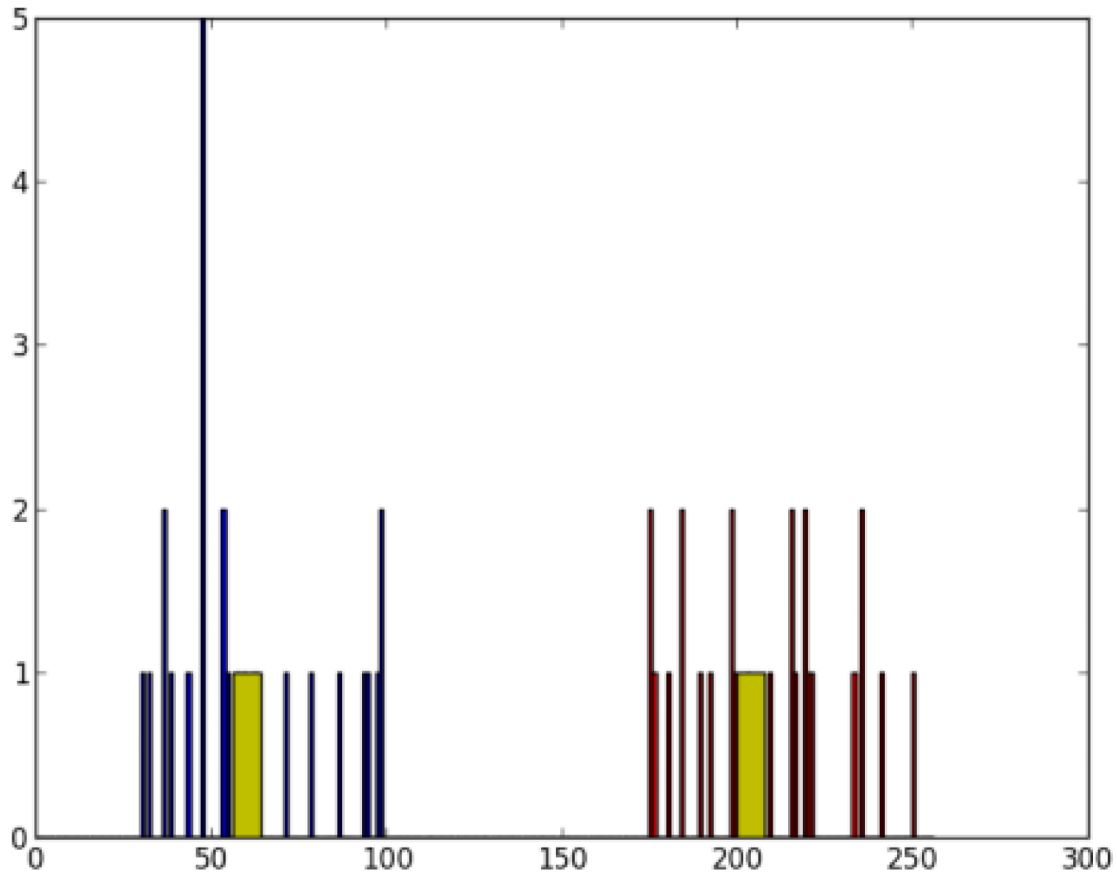
This gives us the compactness, labels and centers. In this case, I got centers as 60 and 207. Labels will have the same size as that of test data where each data will be labelled as '0','1','2' etc. depending on their centroids. Now we split the data to different clusters depending on their labels.

```
A = z[labels==0]
B = z[labels==1]
```

Now we plot A in Red color and B in Blue color and their centroids in Yellow color.

```
# Now plot 'A' in red, 'B' in blue, 'centers' in yellow
plt.hist(A,256,[0,256],color = 'r')
plt.hist(B,256,[0,256],color = 'b')
plt.hist(centers,32,[0,256],color = 'y')
plt.show()
```

Below is the output we got:



2. Data with Multiple Features

In previous example, we took only height for t-shirt problem. Here, we will take both height and weight, ie two features.

Remember, in previous case, we made our data to a single column vector. Each feature is arranged in a column, while each row corresponds to an input test sample.

For example, in this case, we set a test data of size 50x2, which are heights and weights of 50 people. First column corresponds to height of all the 50 people and second column corresponds to their weights. First row contains two elements where first one is the height of first person and second one his weight. Similarly remaining rows corresponds to heights and weights of other people. Check image below:

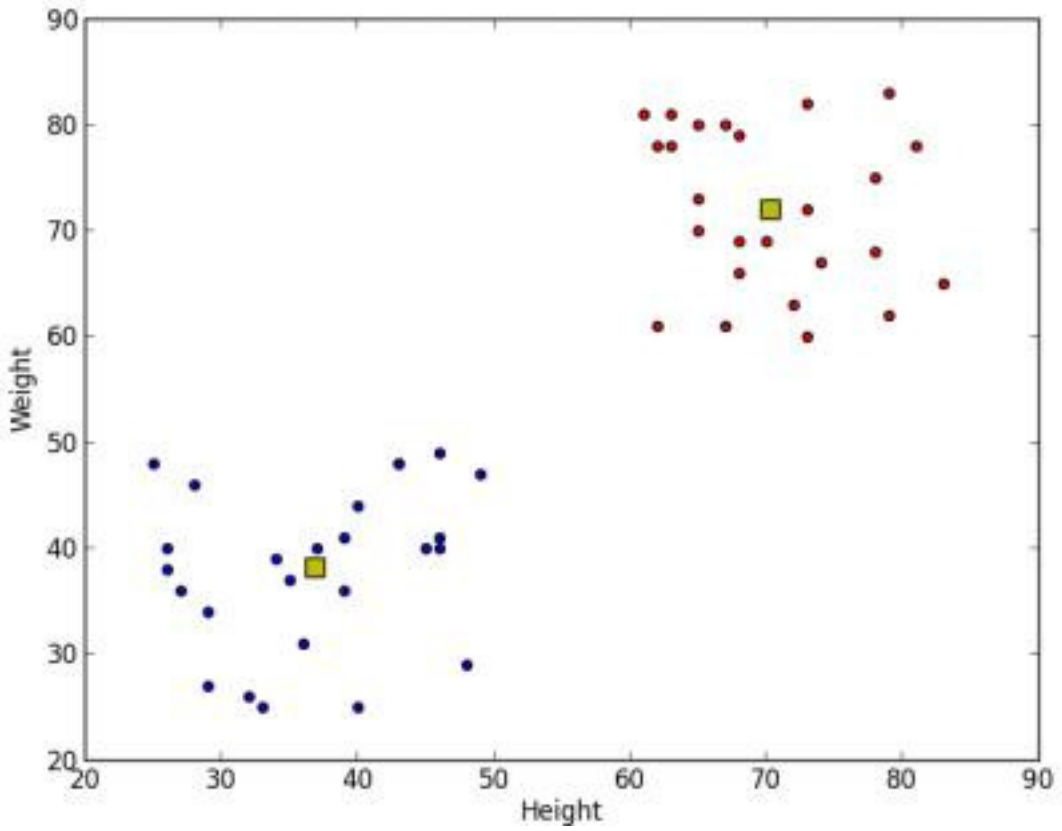
	Features	
	Height	Weight
Person 1	H1	W1
Person 2	H2	W2
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
Person 50	H50	W50

Now I am directly moving to the code:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

X = np.random.randint(25,50,(25,2))
Y = np.random.randint(60,85,(25,2))
Z = np.vstack((X,Y))

# convert to np.float32
Z = np.float32(Z)

# define criteria and apply kmeans()
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
ret,label,center=cv2.kmeans(Z,2,None,criteria,10,cv2.KMEANS_RANDOM_CENTERS)

# Now separate the data, Note the flatten()
A = Z[label.ravel()==0]
B = Z[label.ravel()==1]

# Plot the data
plt.scatter(A[:,0],A[:,1])
plt.scatter(B[:,0],B[:,1],c = 'r')
plt.scatter(center[:,0],center[:,1],s = 80,c = 'y', marker = 's')
plt.xlabel('Height'),plt.ylabel('Weight')
```

```
plt.show()
```

Below is the output we get:



3. Color Quantization

Color Quantization is the process of reducing number of colors in an image. One reason to do so is to reduce the memory. Sometimes, some devices may have limitation such that it can produce only limited number of colors. In those cases also, color quantization is performed. Here we use k-means clustering for color quantization.

There is nothing new to be explained here. There are 3 features, say, R,G,B. So we need to reshape the image to an array of Mx3 size (M is number of pixels in image). And after the clustering, we apply centroid values (it is also R,G,B) to all pixels, such that resulting image will have specified number of colors. And again we need to reshape it back to the shape of original image. Below is the code:

```
import numpy as np
import cv2

img = cv2.imread('home.jpg')
Z = img.reshape((-1,3))

# convert to np.float32
Z = np.float32(Z)

# define criteria, number of clusters(K) and apply kmeans()
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
K = 8
ret,label,center=cv2.kmeans(Z,K,None,criteria,10,cv2.KMEANS_RANDOM_CENTERS)
```

```
# Now convert back into uint8, and make original image
center = np.uint8(center)
res = center[label.flatten()]
res2 = res.reshape((img.shape))

cv2.imshow('res2',res2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

See the result below for K=8:

Additional Resources

Exercises

1.9 Computational Photography

Here you will learn different OpenCV functionalities related to Computational Photography like image denoising etc.

- *Image Denoising*