Addition: K-means in Matlab (2018 version)

In the k-means algorithm, we're interested in getting the cluster centers and distortion error as well as the cluster indices for each input point. Matlab has several versions of calling the function.

The data in matrix X is input as "number of points by dimension". The basic function call is given as the following. The output vector gives the cluster index of each data point.

```
idx=kmeans(X,k);
```

We usually want to track some measures of how good the clustering is, and we'll normally want to store the cluster centers as well. In that case, we can output the cluster centers and the individual distortion errors (DistErr is a vector with k values).

```
[idx,Centers,DistErr]=kmeans(X,k)
```

There are some other interesting additions to the `kmeans` command. As you train, you might notice that the distortion error that you end up with often depends on how the centers are initialized. A good practice is to initialize the algorithm several times so that you can both track the distortion error and choose the best arrangement. Matlab has this built-in as the "Replicates" option. Also, if we want to display the results as we train, we can set that option as well:

```
opts=statset('Display','final');
[idx,C,disterr]=kmeans(X,k,'Replicates',5,'Options',opts)
```

New in Matlab's Algorithm

There are several new additions to the basic algorithm that have been added over the last few years.

- There are two phases now for clustering- The first phase is as described (Matlab calls it batch training), and there is now a second phase (online) that re-assigns each individual point to a new cluster if that re-assignment will lower the distortion error.

  You do get a better clustering in terms of the distortion error, but this can be time consuming if you have a large data set.

  Matlab defaults to the online phase being "off". To turn it "on", we would use the pair: `'Online Phase','on'` in the k-means command. For example,

  ```
  [idx,C,disterr]=kmeans(X,k,'OnlinePhase','on');
  ```

- The "k-means++" algorithm to initialize the cluster centers, and this is now Matlab's default method. The algorithm uses a heuristic to choose cluster centers with certain probabilities:

1. Choose c_1 uniformly at random from the data.
2. Compute distances between the data and c_1, $d(x_m, c_1)$.
3. Now we compute a probability for each $x_m \in X$: Select c_2 at random from X with probability ($m = 1, 2, \cdots, p$)

$$\frac{d^2(x_m, c_1)}{\sum_{j=1}^{p} d^2(x_j, c_1)}$$

4. We'll select each subsequent center from the data with a probability that is proportional to the distance from itself to the closest centroid. For $m = 1, 2, \cdots, p$ and $r = 1, 2, \cdots, k-1$, choose center c_k at random from X with probability:

$$\frac{d^2(x_m, c_r)}{\sum_{\{h: x_h \in c_r\}} d^2(x_h, c_r)}$$

- To start by selecting k points at random from X, we would use the option pair: `'Start','sample'` (the example below uses this option).

Sample Training Session

Using the iris data, we give a short training session below.

```
load fisheriris;
X=meas;  %X is 150 x 4

opts=statset('Display','final');
[idx,C,disterr]=kmeans(X,3,'Replicates',5,'Start','Sample','Options',opts);
plot(idx,'.');
```

Exercise: Image segmentation

In the image segmentation problem, given an image, we want to extract the regions that are distinct colors or objects. Consider the following example, where we examine the petals of a flower sample.

Before we look at the clustering algorithm, it might be helpful to look at the photo and look at how the colors are stored. Be sure you've downloaded the photo from the class website. Click on the "Apps" tab, then find the "Color Thresholder". You'll see an option to "Load Image". Select that, and find the image you downloaded. You should see something like Figure 3. Select the L*a*b color space.

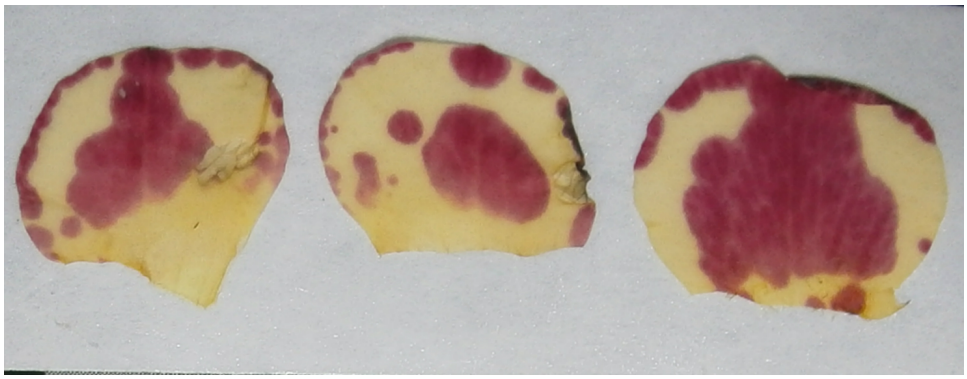

Figure 1: The original image. Our goal is to pull apart the colors: Yellows, reds, versus background.

Finally, we want to use the sliders to the right (see Figure **??**). See if you can isolate the red colors (sample solution shown).

We'll now perform k-means clustering on our image and show that it does a pretty good job. Here are the steps for the **homework**:

1. Load the image, and convert the image from RGB to L*a*b.

```
X=load('P355F1.jpg');     %X is a uint8 matrix
Y=rgb2lab(X);
```

2. Next, strip away the L values- we'll cluster only using "a" and "b" values (the last two "columns" of the third dimension). What would the Matlab command be for that?

3. Currently, we have an array that is $m \times n \times 2$. We will convert that (using the `reshape` command) into a matrix that is $mn \times 2$, then we'll cluster using that matrix. Hint: Keep track of the current values of m and n!

4. Cluster the data, and have Matlab output the cluster indices in an $mn \times 1$ vector.

5. Visualize the clusters by `reshape` back into an $m \times n$ matrix, then use `imagesc` to show the results.

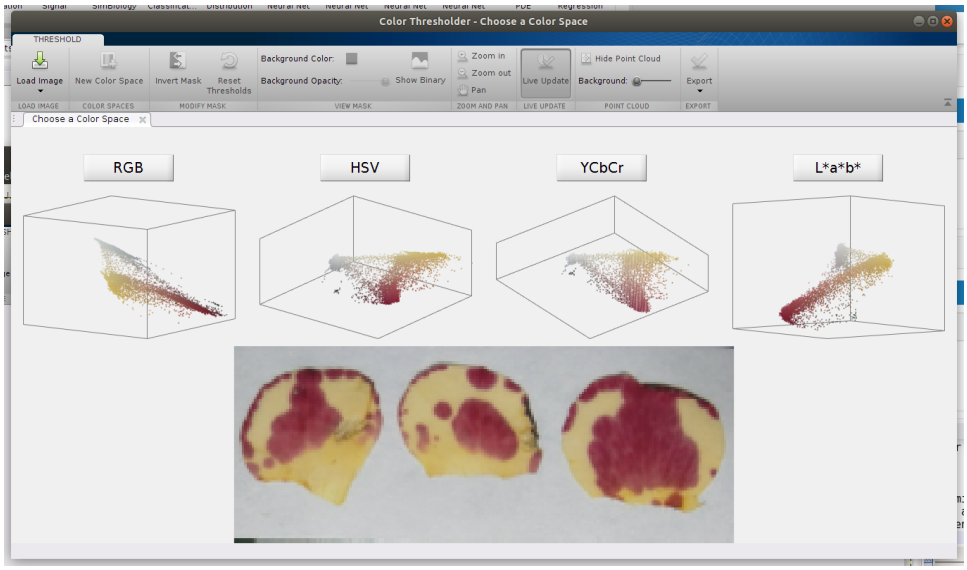

Figure 2: This is the Color Threshold dialog box, with the image loaded. You can try looking at different color representations this way.

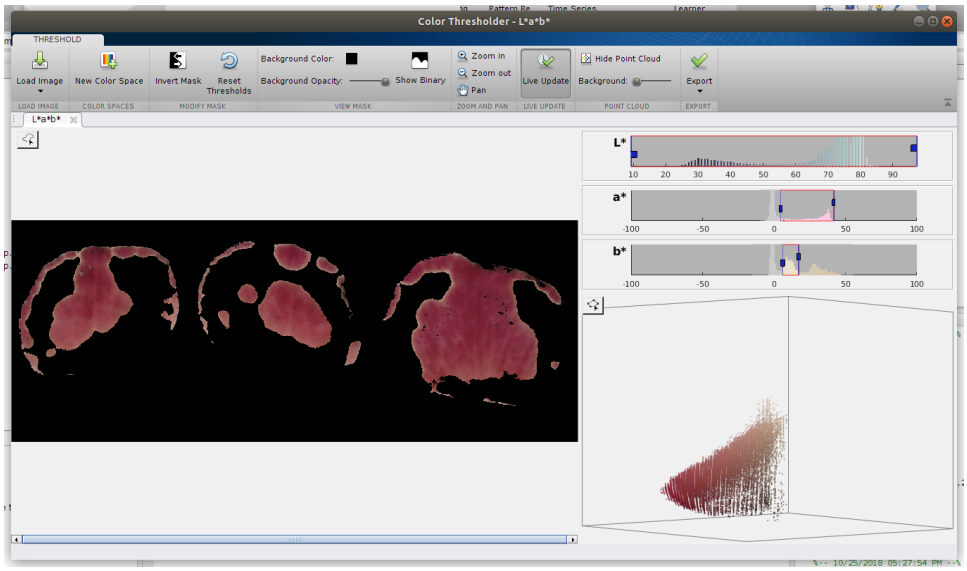

Figure 3: In the L*a*b color space, we can adjust the sliders to see if its possible to isolate the colors. Shown is one possible solution.