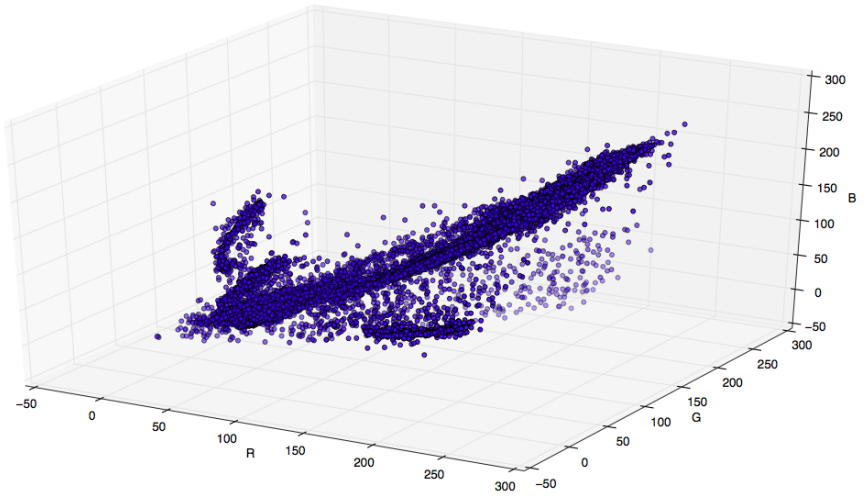
CS 520: Clustering (ft. Luna Rose)

16:198:520
Instructor: Wes Cowan

Consider the following image.

Figure 1: Luna Rose - She's a hero, you see.

The original file is 533?x?960 pixels or $511,680$ pixels in total. Each pixel has color specified by a **(red, blue, green)** value, each value an integer between 0 and 255. This gives a total of $16,777,216$ possible colors, but in fact the picture contains much fewer colors than this - only $61,045$ unique colors. And in fact, these colors are not uniformly distributed - we can generate a plot of the colors used, and see that there is a lot of structure.



Consider the problem of reducing the complexity and size (in memory) of the image, while maintaining the quality as much as possible. One possible approach would be to reduce the number of colors used. One approach might be to 'bin' RGB values, so for instance instead of red taking on any possible value from 0 to 255, we round to a nearest representative, e.g., **red** $\in \{32, 96, 160, 224\}$. In this case, the color $(80, 16, 144)$ would be replaced with $(96, 32, 160)$. In this case, taking four possible choices for each color value, this leads to 64 possible unique color 'bins', 29 of which occur in the image.

Figure 2: Coloring the original image based on the 64 possible color bins.

Obviously, reducing the color space like this captures a lot of the variation and complexity of the original image - but it looks wrong. Part of the issue is that the binning has been done in some sense independent of the way the colors actually occur in the image. We simply partitioned the color space and went with it. If all of the colors had fallen

in the same bin, the reduced image would've come out one uniform color. We can avoid this, and salvage much of the original image, but constructing data-defined bins or clusters.

1 Clustering and k-Means

Suppose that we are given a sequence of data points Data $= \{\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_N\}$ in some space - in the example above, each (3-dimensional) data point represents a color that occurs in the original image, with $N = 511680$.

A k-clustering of the data is a partition of the data into k sets, $C_1, C_2, \ldots, C_k$, where every data point belongs to some cluster. A good clustering should have the property that everything in the cluster is more similar to each other than to anything in a different cluster. To that end, suppose that $\underline{c}_i$ is the 'center' of cluster C_i - ideally we want to achieve a clustering (find clusters $\{C_i\}$ with centers $\{\underline{c}_i\}$ that minimize the following quantity:

$$\sum_{i=1}^{k} \sum_{\underline{x} \in C_i} \text{dist}(\underline{x}, \underline{c}_i), \tag{1}$$

so that every point is closer to the center of its cluster than the center of any other cluster. The inter-cluster variation should be as small as possible.

The following observations lead to a simple algorithm:

- Given any choice of clusters and centers, if $\underline{x}$ is in cluster C_i, but $\text{dist}(\underline{x}, \underline{c}_i) > \text{dist}(\underline{x}, \underline{c}_j)$, the total variation can be reduced by moving $\underline{x}$ from C_i to C_j.
- Given a cluster C_i with 'center' $\underline{c}_i$, the total variation can be reduced by replacing $\underline{c}_i$ with a new center $\underline{c}'_i$ where

$$\underline{c}'_i = \text{argmin}_{\underline{c}} \sum_{\underline{x} \in C_i} \text{dist}(\underline{x}, \underline{c}). \tag{2}$$

This can be systemized, as **Lloyd's k-Means Algorithm**. Taking dist as the square norm:

0) Choose some value of k as the desired number of clusters. Select k 'centers' $\underline{c}_i(0)$ at random, potentially from the data itself.

1) At time t, construct cluster $C_i(t)$ to be the points in Data that are closer to $\underline{c}_i(t)$ than any other $\underline{c}_j(t)$ for $j \neq i$.

2) For each cluster $C_i(t)$, compute a *new* center, averaging the points in the cluster:

$$\underline{c}_i(t+1) = \frac{1}{|C_i(t)|} \sum_{\underline{x} \in C_i(t)} \underline{x}. \tag{3}$$

3) Repeat.

Taking $k = 29$ (so as to use the same number of colors as in the previous example), we can run clustering on the palette of the original image, clustering and recentering. Successive results are shown below, starting with initial (randomized) centers, and adjusting the centers to be more representative of their clusters.

Figure 3: Coloring the original based on $k = 29$ clusters after 0, 1, and 10 rounds of training.

Below, the algorithm is run to full convergence of the clusters (59 rounds), and compared to the original reduced palette. Even with the same number of colors used (29), the clustered image more faithfully reproduces the original image. By defining the 'bins' in terms of the data itself, rather than an arbitrary binning choice, clustering discovers important relationships and structure within the data, and reproduces that in a representative way.

Figure 4: Comparing the original image (center) with the binned color image (left) and k-means run to convergence (right). Clusters converged after 59 rounds.

This is one of the goals of 'unsupervised' learning - the algorithm, driven by the data itself, uncovers structure and relations within the data. This is not only applicable to the problem of faithful data reduction as it was applied here. Clusters might also represent discovered concepts, for instance healthy vs cancerous in a data set of tissue images. Or if each data point represents a set of vital statistics about members of a population, clustering might uncover correlations to do with health, spending habits, finances, or geography. This can be a very powerful tool for data exploration.

Some closing thoughts:

- How long should the algorithm be run? Lloyd's algorithm in particular suffers from some numerical instabilities which can result in very long convergence times. Ideally, the algorithm can be terminated when no more data points need to be moved between clusters and as a result the recenterings do not move the centers. At this point, every point is closer to the center of its cluster than the center of any other cluster, and no more updating is needed. But useful results can frequently be achieved long before this - even a sub-optimal clustering may be highly representative of the data. Other algorithms exist for improving convergence and initialization, but there is added computational overhead to implement them.
- The whole point of unsupervised learning is to minimize the amount of subjective decisions that must be made. However, two things in particular stand out here as needing to be defined for your algorithm:
  - **Distance:** Given a data point and a center, how should the distance between them be defined? One possibility is to simply take some kind of vector norm - but this implicitly equates the importance of all components, which may not be the case. When it comes to human perception, for instance, differing amounts of green can make a much stronger difference to perceived color than differing amounts of red.

A number of schemes have been derived to try to quantify this, but for the above experiment I utilized the following notion of *color difference*, weighting the contributions of each color to the total distance.

$$\text{dist}(\text{color}_1, \text{color}_2) = \sqrt{2(r_1 - r_2)^2 + 4(g_1 - g_2)^2 + 3(b_1 - b_2)^2}. \tag{4}$$

Having some notion of your data, what is relevant in your data to the problem you are trying to solve, can help direct a notion of comparison and distance, and improve your clustering algorithm.

- **Number of Clusters k:** The most significant choice that needs to be made, however, is the number of clusters to be used. In the above case, I used $k = 29$ - but this choice was arbitrarily driven by the number of colors that occurred in the original naive binning approach, rather than driven by the data itself. In general, it can be very difficult to know in advance how many clusters may be appropriate (especially with high dimensional data that may not be easily visualized). To that end, the best approach is frequently experimentation. Cluster with various values of k and note how the over all clustering loss performs: for too many clusters, every data point will effectively get its own cluster, and the total loss will be near 0; for too few clusters, the loss will be quite high as disparate data points will end up in the same cluster. As k increases, there will generally be a significant drop in loss before the loss begins to plateau and trail off to 0 - a practical choice for k value is frequently near this 'elbow', marking the transition between these two regimes.

- In a similar vein, it may be useful to do some pre-processing on your data. This is connected to the notion of how you define distance. For instance, if the components of your data point represent incredibly disparate things (average length of grass locally, average volume of rain fall, number of packages delivered in the holiday season), some pre-processing such as rescaling or weighting may be needed to render these things comparable. Do not let your comparisons be dominated by one aspect of your data because it is in different units than another (inches vs miles, etc.).