K-Means Clustering and Related Algorithms

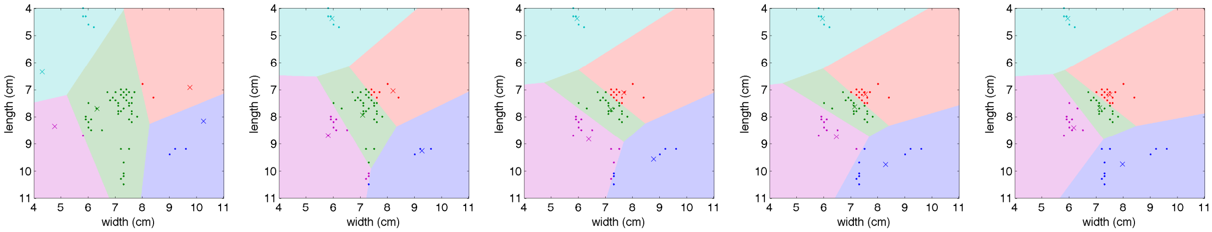
Ryan P. Adams
COS 324 – Elements of Machine Learning
Princeton University

In its broadest definition, machine learning is about automatically discovering structure in data. Those data can take many forms, depending on what our goals are. The data might be something like images and labels, in a supervised visual object recognition task, or the might be something much more abstract, such as the "life experiences" of a robot embodied in the world. Here we will examine one of the simplest ideas for discovering structure: we will look for groups, or **clusters** among the data. Intuitively, a cluster is a subset of data in which the data are in some sense more similar to each other than they are to data not in the cluster. For example, in a large set of news articles, one cluster might correspond to a group of documents that are about baseball, because these will tend to use similar words and be very different from documents about, say, international politics, which might form a different cluster.

Finding groups in data is just one motivation for clustering. We might also imagine prediction tasks based on these groups. For example, if we had a data set with images of different kinds of animals, we might hope that a clustering algorithm would discover the animal types and be able to answer questions such as "is the animal in image A of the same type as the animal in image B?" We can also motivate clustering algorithms by thinking about compression. If I had to summarize a large data set by a small set of examples, it might be sensible to choose those examples to represent different distinct groups in the data. When each of the data is well described by its associated example, then I might believe I had discovered some interesting structure. It can also be useful, as we'll see later in the course, to think of clustering algorithms as providing **features** of the data that summarize a lot of information in the data in a concise way. Good features are critical for supervised learning tasks to perform well (Coates et al., 2011). Even simple clustering algorithms can discover important information about, for example, whether a recorded speech signal has a male or female speaker. Finally, we can also view clustering algorithms as a funny kind of classification in which we have to discover the labels for ourselves. That is, maybe we have a big bag of images and we don't *a priori* have concepts such as "horse" or "dog" and so we have to discover these (hopefully) coherent groups from scratch.

1 Clustering

For most clustering algorithms, the main thing that we need is a notion of distance between our data. If our data live in some space $\mathcal{X}$, then we need a function that takes two points in $\mathcal{X}$, say, $\boldsymbol{x}$



(a) Initialization (b) Iteration 1 (c) Iteration 2 (d) Iteration 3 (e) Iteration 4

Figure 1: Four iterations of K-Means applied to the lengths and widths of fruit (oranges and lemons), as measured in centimeters. Here, $K = 5$ and the initial means were chosen randomly in the square shown. The colored regions show the Voronoi partitions induced by each cluster center, and the data are colored according to their association. The cluster centers are shown as ×.

and $\boldsymbol{x}'$ and computes a distance between them. We'll write such a distance as $||\boldsymbol{x} - \boldsymbol{x}'||$. If $\mathcal{X}$ is $\mathbb{R}^D$, then a natural choice would be the Euclidean (L^2) distance we've studied this term in other contexts:

$$||\boldsymbol{x} - \boldsymbol{x}'||_2 = \sqrt{\sum_{d=1}^{D}(x_d - x'_d)^2}. \tag{1}$$

There are many distance metrics that one might come up with, depending on what your data are and what "similarity" means for the problem you want to solve. For strings or DNA sequences, one might use edit distance.[1] For bit vectors, it might be sensible to use Hamming distance.[2] This choice is important because it will determine whether two objects should want to be in the same group or not.

You also need to decide is how many groups you want. This is the number K that gives the K-means algorithm its name. Choosing K can be a bit of an art, because it depends on what kind of structure you're looking for. Sometimes you might know how many clusters there are in advance. Other times, you might want to "oversegment" or "undersegment" the data, depending on whether you'd like to end up with many clusters or just a few. In the compression view of clustering, this boils down to asking whether you'd like a more compressed representation that loses information (smaller K), or a less compressed representation that keeps more information about your data (larger K). If you're using K-Means for learning a feature representation, it's usually a good idea to use a larger K. If you want to interpret the groups, then perhaps you want to go with a smaller K.

Our data are N points in $\mathcal{X}$. Let's denote the nth of these as $\boldsymbol{x}_n$, so we can write the data as the set $\{\boldsymbol{x}_n\}_{n=1}^N$. Clustering algorithms assign every one of these data to one of the K clusters. What we're doing is trying to find a good (ideally, the best) assignment of the data to the clusters. We represent these assignments by giving every one of the N data a binary **responsibility vector $\boldsymbol{r}_n$**. This vector is all zeros except in one component, which corresponds to the cluster it is assigned to. That is, if $\boldsymbol{x}_n$ is assigned to cluster k, then $r_{nk} = 1$ and all of the other entries in $\boldsymbol{r}_n$ are zero. This

[1] http://en.wikipedia.org/wiki/Edit_distance

[2] http://en.wikipedia.org/wiki/Hamming_distance

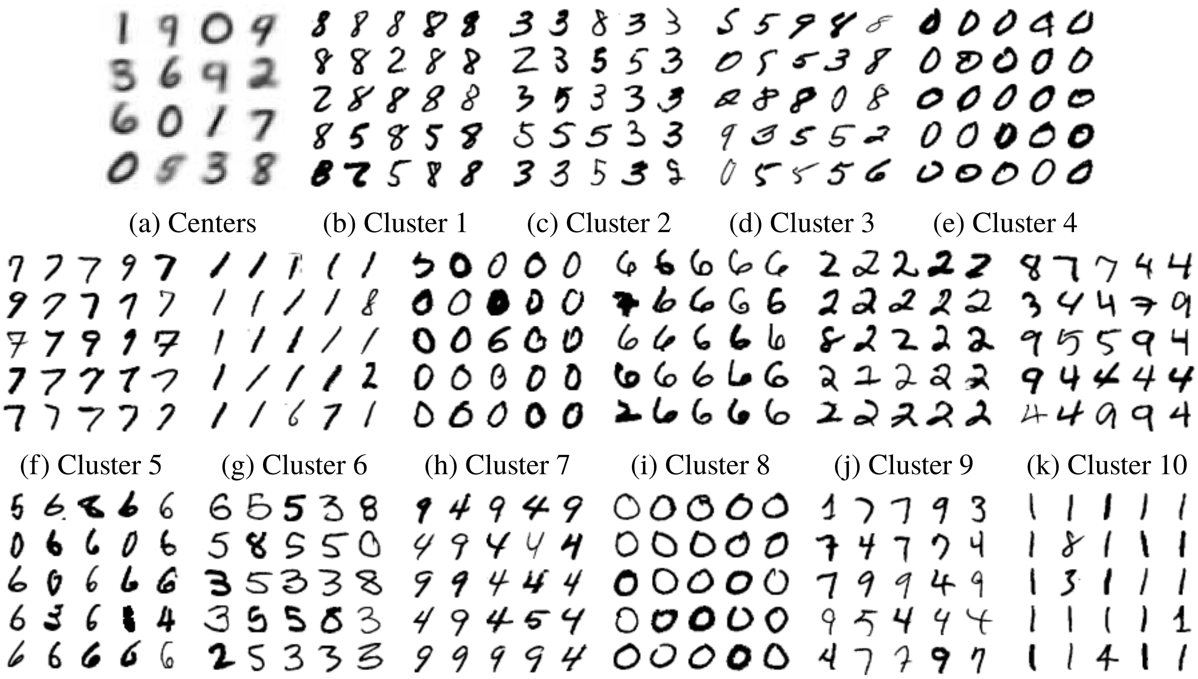
Algorithm 1 K-Means Clustering (Lloyd's Algorithm) *Note: written for clarity, not efficiency.*

```
Input: Data vectors $\{\boldsymbol{x}_n\}_{n=1}^N$, number of clusters $K$
for $n \leftarrow 1 \dots N$ do                                   ▷ Initialize all of the responsibilities.
    $\boldsymbol{r}_n \leftarrow [0, 0, \cdots, 0]$                ▷ Zero out the responsibilities.
    $k' \leftarrow$ RandomInteger$(1, K)$                          ▷ Make one of them randomly one to initialize.
    $r_{nk'} = 1$
end for
repeat
    for $k \leftarrow 1 \dots K$ do                                ▷ Loop over the clusters.
        $N_k \leftarrow \sum_{n=1}^N r_{nk}$                       ▷ Compute the number assigned to cluster $k$.
        $\boldsymbol{\mu}_k \leftarrow \frac{1}{N_k} \sum_{n=1}^N r_{nk} \boldsymbol{x}_n$   ▷ Compute the mean of the $k$th cluster.
    end for
    for $n \leftarrow 1 \dots N$ do                                ▷ Loop over the data.
        $\boldsymbol{r}_n \leftarrow [0, 0, \cdots, 0]$            ▷ Zero out the responsibilities.
        $k' \leftarrow \arg\min_k ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2$   ▷ Find the closest mean.
        $r_{nk'} = 1$
    end for
until none of the $\boldsymbol{r}_n$ change
Return assignments $\{\boldsymbol{r}_n\}_{n=1}^N$ for each datum, and cluster means $\{\boldsymbol{\mu}_k\}_{k=1}^K$.
```

is an example of **one-hot coding** in which an integer between 1 and K is encoded as a length-K binary vector that is zero everywhere except for one place.

2 The K-Means Algorithm

When the data space $\mathcal{X}$ is $\mathbb{R}^D$ and we're using Euclidean distance, we can represent each cluster by the point in data space that is the average of the data assigned to it. Since each cluster is represented by an average, this approach is called *K-Means*. The K-Means procedure is among the most popular machine learning algorithms, due to its simplicity and interpretability. Pseudocode for K-Means is shown in Algorithm 1. K-means is an iterative algorithm that loops until it converges to a (locally optimal) solution. Within each loop, it makes two kinds of updates: it loops over the responsibility vectors $\boldsymbol{r}_n$ and changes them to point to the closest cluster, and it loops over the **mean vectors** $\boldsymbol{\mu}_k$ and changes them to be the mean of the data that currently belong to it. There are K of these mean vectors (hence the name of the algorithm) and you can think of them as "prototypes" that describe each of the clusters. The basic idea is to find a prototype that describes a group in the data and to use the $\boldsymbol{r}_n$ to assign the data to the best one. In the compression view of K-Means, you can think of replacing your actual datum $\boldsymbol{x}_n$ with its prototype and then trying to find a situation in which that doesn't seem so bad, i.e., that compression will not lose too much information if the prototype accurately reflects the group. When updating the assignments, we tend to use the squared distance, rather than the actual distance, as this doesn't change the answer and we avoid the square root in Eq. 1.



(a) Centers (b) Cluster 1 (c) Cluster 2 (d) Cluster 3 (e) Cluster 4

(f) Cluster 5 (g) Cluster 6 (h) Cluster 7 (i) Cluster 8 (j) Cluster 9 (k) Cluster 10

(l) Cluster 11 (m) Cluster 12 (n) Cluster 13 (o) Cluster 14 (p) Cluster 15 (q) Cluster 16

Figure 2: This is the result of K-Means clustering applied to the MNIST digits data. (a) The 16 cluster centers. (b-q) 25 data examples are shown for each of the 16 clusters. The clusters roughly grab digits with similar stroke patterns.

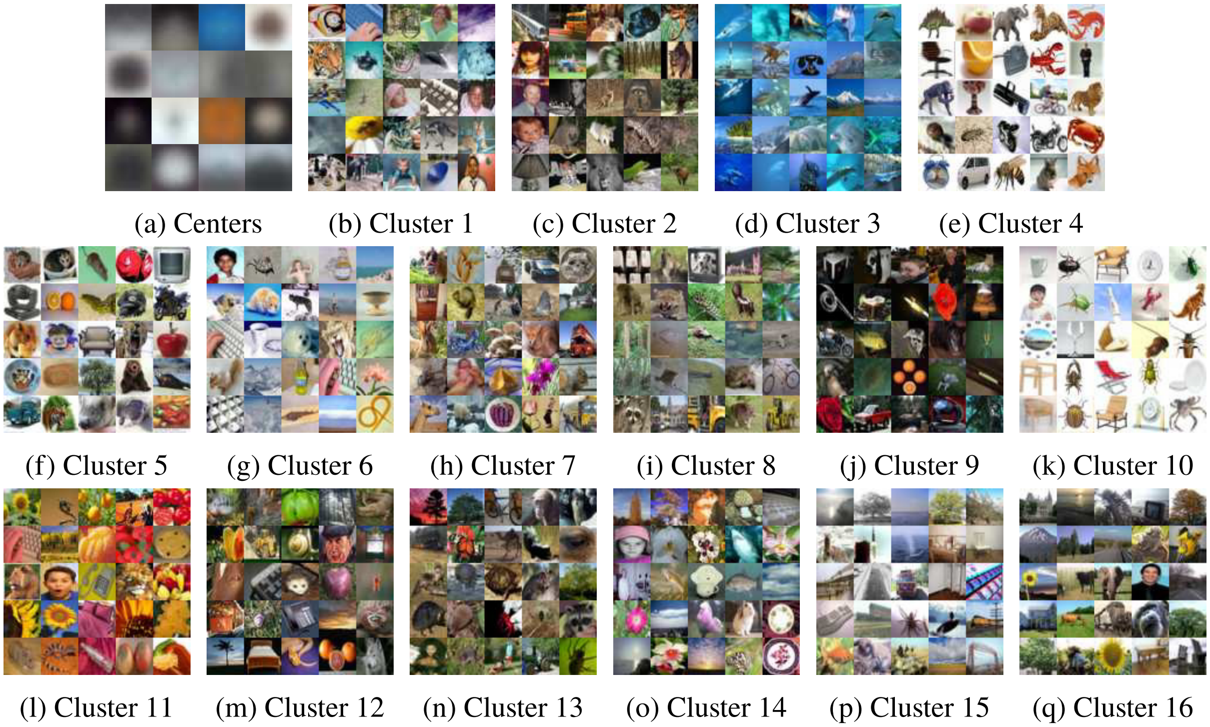
Figure 1 shows several iterations of the K-Means clustering algorithm applied to two-dimensional data. These are the lengths and widths of fruit (oranges and lemons) purchased by Iain Murray.[3] These distances are in centimeters. I initialized the centers randomly within the square shown, and used $K = 5$.

2.1 Example: Handwritten Digits

Figure 2 shows the result of applying K-Means clustering to the MNIST handwritten digits.[4] There are 60000 digits and each is a 28×28 grayscale image, i.e., each pixel is an unsigned byte between 0 and 255. I loaded the data into Matlab, turned it into a big 60000×784 matrix, casted it into a double and then divided by 255. This gave me a bunch of vectors between 0 and 1. I then initialized with K-Means++, followed by Lloyd's algorithm. I did this with $K = 16$. I used the vectorization tricks that I mention here, and on my laptop it converged in less than a minute.

[3] http://homepages.inf.ed.ac.uk/imurray2/teaching/oranges_and_lemons/

[4] http://yann.lecun.com/exdb/mnist/



(a) Centers (b) Cluster 1 (c) Cluster 2 (d) Cluster 3 (e) Cluster 4

(f) Cluster 5 (g) Cluster 6 (h) Cluster 7 (i) Cluster 8 (j) Cluster 9 (k) Cluster 10

(l) Cluster 11 (m) Cluster 12 (n) Cluster 13 (o) Cluster 14 (p) Cluster 15 (q) Cluster 16

Figure 3: This is the result of K-Means clustering applied to the CIFAR-100 image data. (a) The 16 cluster centers. (b-q) 25 data examples are shown for each of the 16 clusters. The clusters primarily pick up on low-frequency color variations.

2.2 Example: Color Images

Figure 3 shows the result of applying K-Means clustering to the CIFAR-100 color images.[5] There are 50000 32×32 color images, i.e., each pixel is an RGB triplet of unsigned bytes between 0 and 255. I loaded the data into Matlab, turned it into a big 50000×3072 matrix, casted it into a double and then divided by 255. This gave me a bunch of vectors between 0 and 1. I then initialized with K-Means++, followed by Lloyd's algorithm. I did this with $K = 16$. I used the vectorization tricks that I mention here, and on my laptop it converged in less than three minutes.

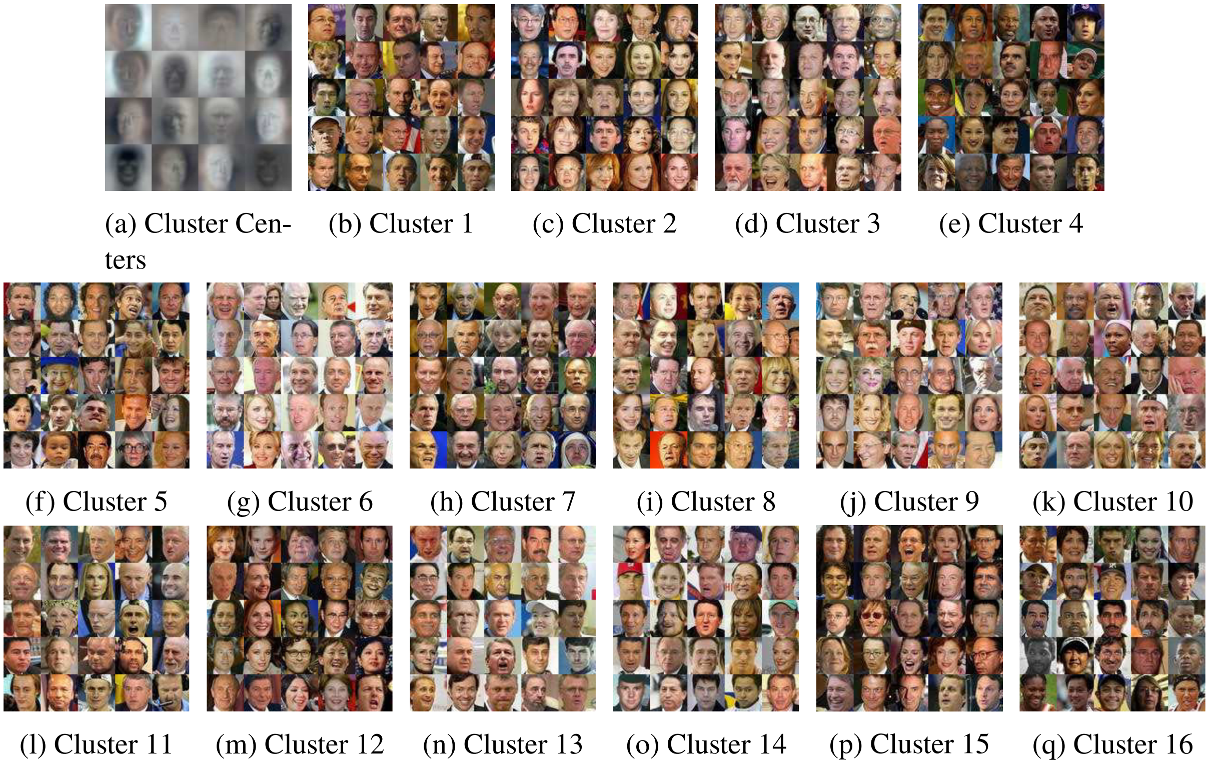
2.3 Example: Images of Faces

Figure 4 shows the result of applying K-Means clustering to a preprocessed variant of the Labeled Faces in the Wild data.[6] There are 13233 40×40 color images, i.e., each pixel is an RGB triplet of unsigned bytes between 0 and 255. I used the aligned variant of the data, cropped out the middle 150 pixel square, and resized the cropped image to 40×40 pixesl. I used Imagemagick[7] to do this quickly, as it provides a very nice command line tool for image processing. I loaded the image data

[5] `http://www.cs.toronto.edu/~kriz/cifar.html`

[6] `http://vis-www.cs.umass.edu/lfw/`

[7] `http://www.imagemagick.org/`



(a) Cluster Centers (b) Cluster 1 (c) Cluster 2 (d) Cluster 3 (e) Cluster 4

(f) Cluster 5 (g) Cluster 6 (h) Cluster 7 (i) Cluster 8 (j) Cluster 9 (k) Cluster 10

(l) Cluster 11 (m) Cluster 12 (n) Cluster 13 (o) Cluster 14 (p) Cluster 15 (q) Cluster 16

Figure 4: This is the result of K-Means clustering applied to a preprocessed variant of the Labeled Faces in the Wild image data. (a) The 16 cluster centers. (b-q) 25 data examples are shown for each of the 16 clusters. The clusters capture a combination of face shape, background, and illumination.

into Matlab, turned it into a big 13233 × 4800 matrix, casted it into a double and then divided by 255. This gave me a bunch of vectors between 0 and 1, which I standardized. I then initialized with K-Means++, followed by Lloyd's algorithm. I did this with $K = 16$. I used the vectorization tricks that I mention here, and on my laptop it converged in less than three minutes.

2.4 Example: Grolier Encyclopedia Articles

Table 1 shows the result of applying K-Means to a set of 30991 articles from Grolier's Encyclopedia.[8] The articles are represented as a sparse vector of word counts with a vocabulary of the 15276 most common words, excluding *stop words*[9] such as "the" and "and". These counts were treated as the features directly, leading to a very simple notion of distance. I used $K = 12$ and initialized with K-Means++ before applying Lloyd's algorithm.

[8] `http://www.cs.nyu.edu/~roweis/data.html`

[9] `http://en.wikipedia.org/wiki/Stop_words`

Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
education	south	war	war	art	light
united	population	german	government	century	energy
american	north	british	law	architecture	atoms
public	major	united	political	style	theory
world	west	president	power	painting	stars
social	mi	power	united	period	chemical
government	km	government	party	sculpture	elements
century	sq	army	world	form	electrons
schools	deg	germany	century	artists	hydrogen
countries	river	congress	military	forms	carbon

Cluster 7	Cluster 8	Cluster 9	Cluster 10	Cluster 11	Cluster 12
energy	god	century	city	population	cells
system	world	world	american	major	body
radio	century	water	century	km	blood
space	religion	called	world	mi	species
power	jesus	time	war	government	cell
systems	religious	system	john	deg	called
television	steel	form	life	sq	plants
water	philosophy	united	united	north	animals
solar	science	example	family	south	system
signal	history	life	called	country	human

Table 1: This is the result of a simple application of K-Means clustering to a set of Grolier encyclopedia articles. Shown above are the words with the highest "mean counts" in each of the cluster centers, with clusters as columns. Even with this very simple approach, K-Means identifies groups of words that seem conceptually related.

3 Derivation

Where does this algorithm come from and what does it do? As with many machine learning algorithms, we begin by defining a loss function which specifies what solutions are good and bad. This loss function takes as arguments the two sets of parameters that we introduced in the previous sections, the responsibilities $\boldsymbol{r}_n$ and the means $\boldsymbol{\mu}_k$. Given these parameters and the data vectors $\boldsymbol{x}_n$, what does it mean to be in a good configuration versus a bad one? One intuition is that good settings of $\boldsymbol{r}_n$ and $\boldsymbol{\mu}_k$ will be those in which as many of the data as possible can be near their assigned $\boldsymbol{\mu}_k$. This fits well with the compression view of K-Means: if each of the $\boldsymbol{x}_n$ was replaced by the appropriate $\boldsymbol{\mu}_k$, then better solutions will have this error be very small on average. We write this as an objective function in terms of the $\boldsymbol{r}_n$ and $\boldsymbol{\mu}_k$:

$$J(\{\boldsymbol{r}_n\}_{n=1}^N, \{\boldsymbol{\mu}_k\}_{k=1}^K) = \sum_{n=1}^N \sum_{k=1}^K r_{nk}||\boldsymbol{x}_n - \boldsymbol{\mu}_k||_2^2. \tag{2}$$

Here we're continuing with the assumption that the distance is Euclidean. This function sums up the squared distances between each example and the prototype it belongs to. The K-Means algorithm minimizes this via coordinate descent, i.e., it alternates between 1) minimizing each of the $\boldsymbol{r}_n$, and 2) minimizing each of the $\boldsymbol{\mu}_k$.

Minimizing the $\boldsymbol{r}_n$ If we look at the sum in Eq. 2, we see that the $\boldsymbol{r}_n$ only appears in one of the outer sums, because it only affects one of the data examples. There are only K possible values for $\boldsymbol{r}_n$ and so we can minimize it (holding everything else fixed) by choosing $r_{nk} = 1$ for the cluster that has the smallest distance:

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_{k'} ||\boldsymbol{x}_n - \boldsymbol{\mu}_{k'}||_2^2 \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Minimizing the $\boldsymbol{\mu}_k$ Having fixed everything else, we note that each $\boldsymbol{\mu}_k$ only depends on one of the parts of the inner sums. We can think about the objective function written in terms of only one of these:

$$J(\boldsymbol{\mu}_k) = \sum_{n=1}^N r_{nk}||\boldsymbol{x}_n - \boldsymbol{\mu}_k||_2^2 \tag{4}$$

$$= \sum_{n=1}^N r_{nk}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathsf{T}}(\boldsymbol{x}_n - \boldsymbol{\mu}_k). \tag{5}$$

Here I've written out the squared Euclidean distance as a quadratic form, because it makes the calculus a bit easier to see. To minimize, we differentiate this objective with respect to $\boldsymbol{\mu}_k$, set the resulting gradient to zero, and then solve for $\boldsymbol{\mu}_k$.

$$\nabla_{\boldsymbol{\mu}_k} J(\boldsymbol{\mu}_k) = \nabla_{\boldsymbol{\mu}_k} \sum_{n=1}^N r_{nk}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathsf{T}}(\boldsymbol{x}_n - \boldsymbol{\mu}_k) \tag{6}$$

$$= \sum_{n=1}^N r_{nk} \nabla_{\boldsymbol{\mu}_k}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathsf{T}}(\boldsymbol{x}_n - \boldsymbol{\mu}_k). \tag{7}$$

Recall that $\nabla_{\boldsymbol{a}} \boldsymbol{a}^{\mathsf{T}} \boldsymbol{a} = 2\boldsymbol{a}$, and we apply the chain rule:

$$\nabla_{\boldsymbol{\mu}_k} J(\boldsymbol{\mu}_k) = -2 \sum_{n=1}^{N} r_{nk}(\boldsymbol{x}_n - \boldsymbol{\mu}_k) = 0 \tag{8}$$

$$\sum_{n=1}^{N} r_{nk} \boldsymbol{x}_n = \boldsymbol{\mu}_k \sum_{n=1}^{N} r_{nk} \tag{9}$$

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} r_{nk} \boldsymbol{x}_n}{\sum_{n=1}^{N} r_{nk}}. \tag{10}$$

Thus, for Euclidean distances anyway, taking the average of the assigned data gives the mean that minimizes the distortion (holding everything else fixed). It can also useful to think of the K-Means clustering algorithm as finding a **Voronoi partition**[10] of the data space.

4 Practical Considerations

4.1 Hardness and Initialization

The objective function in Eq. 2 is highly non-convex, with many local minima. Some of them are very easy to see. For example, you could clearly permute the indices of the clusters and wind up in a “different” solution that was just as good. Coordinate descent, as described here, therefore only finds a local minimum of the objective. Strictly speaking, “K-Means” is not an algorithm, but a problem specified by finding a configuration of the $\boldsymbol{r}_n$ that minimizes Eq. 2 – note that the $\boldsymbol{\mu}_k$ are completely determined by the $\boldsymbol{r}_n$ for a given data set. The iterative algorithm described here is often called Lloyd’s algorithm (Lloyd, 1982), but it represents just one way to optimize the K-Means objective. It turns out that finding (one of) the globally optimal solutions to the K-Means problem is NP-hard (Aloise et al., 2009), even if there are only two clusters.

When faced with highly non-convex optimization problems, a common strategy is to use **random restarts** to try to find a good solution. That is, running Algorithm 1 several times (e.g., 10 or 20), with different random seeds so that the initial $\boldsymbol{r}_n$ might land in better places. Then one looks at the final value of the objective in Eq. 2 to choose the best solution. Another practical strategy for larger data sets is to do these restarts with a smaller subset of the data in order to find some reasonable cluster centers before running the full iteration.

More recently, an algorithm has been proposed that is a bounded-error approximation to the solution of K-Means (Arthur and Vassilvitskii, 2007). This algorithm, called **K-Means++**, is shown in pseudocode in Algorithm 2, and can be an excellent alternative to the simple random initialization shown in Algorithm 1. In fact, Arthur and Vassilvitskii (2007) show that K-Means++ can do well even without using Lloyd’s algorithm at all.

[10] `http://en.wikipedia.org/wiki/Voronoi_diagram`

Algorithm 2 K-Means++ *Note: written for clarity, not efficiency.*

```
**Input:** Data vectors $\{\boldsymbol{x}_n\}_{n=1}^N$, number of clusters $K$
$n \leftarrow \text{RandomInteger}(1, N)$                          ▷ Choose a datum at random.
$\boldsymbol{\mu}_1 \leftarrow \boldsymbol{x}_n$                   ▷ Make this random datum the first cluster center.
**for** $k \leftarrow 2 \dots K$ **do**                            ▷ Loop over the rest of the centers.
    **for** $n \leftarrow 1 \dots N$ **do**                        ▷ Loop over the data.
        $d_n \leftarrow \min_{k'<k} ||\boldsymbol{x}_n - \boldsymbol{\mu}_{k'}||_2$   ▷ Compute the distance to the closest center.
    **end for**
    **for** $n \leftarrow 1 \dots N$ **do**                        ▷ Loop over the data again.
        $p_n \leftarrow d_n^2 / \sum_{n'} d_{n'}^2$                ▷ Compute a distribution proportional to $d_n^2$.
    **end for**
    $n \leftarrow \text{Discrete}(p_1, p_2, \dots, p_N)$           ▷ Draw a datum from this distribution.
    $\boldsymbol{\mu}_k \leftarrow \boldsymbol{x}_n$               ▷ Make this datum the next center.
**end for**
**Return** cluster means $\{\boldsymbol{\mu}_k\}_{k=1}^K$.
```

4.2 Standardizing Data

It is necessary to think carefully about what distances mean in the context of K-Means. For example, imagine that we wanted to cluster cars using two quantities: wheelbase (a length) and performance (horsepower). Consumer cars range in horsepower from about 150hp to perhaps 400hp for sports cars; exotics such as the Bugatti Veyron might go up to 1000hp. Wheelbase (the distance between the front and rear centers of the wheels) we might reasonably measure in meters, typically between 2.5m and 3.5m for passenger cars. If we use these numbers as our inputs, we might get into trouble because the horsepower will dominate the distance, as it is two orders of magnitude larger in scale. This is very unsatisfying because these are just arbitrary units; we could've just as easily used millimeters for wheelbase and then it would dominate over horsepower! A typical way to deal with this is to "standardize" each of the dimensions by finding the transformation so that each dimension has mean zero and standard deviation one. This causes each of the features to be centered at zero and have approximately the same scale (as determined by the data). Algorithm 3 shows such a standardization.

4.3 Missing Data

In practical data analysis problems, we often have missing data. That is, some dimensions of some of the data may be completely missing. How do we deal with this in K-Means? There are a variety of possibilities, but the two main options are *imputation* and *marginalization*. Imputation uses a procedure for inventing the missing data. For example, we might replace the missing data with the mean of the remaining data (if we standardized, this will be zero), or we might regress the missing values on the other features. Imputation can be useful, but it can also be dangerous if it ends up influencing the conclusions that you draw from the data. Marginalization is a more principled approach to dealing with missing data, in which we try to account for all possible values

Algorithm 3 Data Standardization *Note: written for clarity, not efficiency.*

```
**Input:** Data vectors $\{\boldsymbol{x}_n\}_{n=1}^N$
**for** $d \leftarrow 1 \dots D$ **do**                          ▷ Loop over dimensions.
    $m_1 \leftarrow 0$                                      ▷ For storing the total of the values.
    $m_2 \leftarrow 0$                                      ▷ For storing the total of the squared values.
    **for** $n \leftarrow 1 \dots N$ **do**                  ▷ Loop over the data.
        $m_1 \leftarrow m_1 + x_{n,d}$
        $m_2 \leftarrow m_2 + x_{n,d}^2$
    **end for**
    $\mu \leftarrow m_1/N$                                  ▷ Compute sample mean.
    $\sigma^2 \leftarrow (m_2/N) - \mu^2$                   ▷ Compute sample variance.
    **for** $n \leftarrow 1 \dots N$ **do**                  ▷ Loop over the data again to modify.
        $x'_{n,d} \leftarrow (x_{n,d} - \mu)/\sigma$        ▷ Shift by mean, scale by standard deviation.
    **end for**
**end for**
**Return** transformed data $\{\boldsymbol{x}'_n\}_{n=1}^N$
```

of the unknown dimension. That is, we could compute the expectation of the distance between two points, integrating over the missing values. If we've standardized the data (and there aren't too many missing values) then we might reasonably assume that the missing data have a $\mathcal{N}(0,1)$ distribution. The expectation of the squared Euclidean distance, assuming that the dth dimension is missing, is

$$\mathbb{E}\left[\,||\boldsymbol{x} - \boldsymbol{\mu}||_2^2\,\right] = \int_{-\infty}^{\infty} \mathcal{N}(x_d \,|\, 0, 1) \sum_{d'=1}^{D} (x_{d'} - \mu_{d'})^2 \, \mathrm{d}x_d \tag{11}$$

$$= \left[\sum_{d' \neq d} (x_{d'} - \mu_{d'})^2\right] + \int_{-\infty}^{\infty} \mathcal{N}(x_d \,|\, 0, 1)(x_d - \mu_d)^2 \, \mathrm{d}x_d \tag{12}$$

$$= \left[\sum_{d' \neq d} (x_{d'} - \mu_{d'})^2\right] + 1 + \mu_d^2. \tag{13}$$

Basically, this simple marginalization results in us adding a bit to the distance.

4.4 Vectorizing Code

As with almost all the numeric code you write to do data analysis, if you're using a high-level language like Matlab or Python, you'll want to vectorize things as much as possible. That means that rather than writing the simple loops such as I show in these algorithm boxes, you'll want to frame things in terms of operations on vectors and matrices. For example, instead of writing the loops of Algorithm 3, you might write a Python function that takes advantage of vectorized built-ins as in Listing 1. There are no loops in sight, which means that they are (hopefully) being

Listing 1: Python function for standardizing data

```
import numpy as np
def standardize(data):
    '''Take an NxD numpy matrix as input and return a standardized version.'''
    mean = np.mean(data, axis=0)
    std  = np.std(data, axis=0)
    return (data - mean)/std
```

performed in the underlying Fortran or C libraries. These computations can be done very quickly using BLAS[11], for example.

Another very useful trick to know is for computing distances. When the data are one-dimensional, it seems pretty easy. However, with higher dimensional data, it's less obvious how to compute a matrix of distances without looping. Let's imagine that we have an $N \times D$ matrix $\boldsymbol{X}$ and an $M \times D$ matrix $\boldsymbol{Y}$ and that we want to compute an $N \times M$ matrix $\boldsymbol{D}$ with all of the **squared** distances. One entry in this matrix is given by

$$\begin{aligned} D_{n,m} &= (\boldsymbol{x}_n - \boldsymbol{y}_m)(\boldsymbol{x}_n - \boldsymbol{y}_m)^{\mathsf{T}} && (14) \\ &= \boldsymbol{x}_n \boldsymbol{x}_n^{\mathsf{T}} - 2\boldsymbol{x}_n \boldsymbol{y}_m^{\mathsf{T}} + \boldsymbol{y}_m \boldsymbol{y}_m^{\mathsf{T}}, && (15) \end{aligned}$$

where $\boldsymbol{x}_n \in \mathbb{R}^D$ is the nth row of $\boldsymbol{X}$ and $\boldsymbol{y}_m \in \mathbb{R}^D$ is the mth row of $\boldsymbol{Y}$. These are row vectors, so this is a sum of inner products. Using broadcasting, which numpy does naturally and can be done in Matlab using `bsxfun`, this gives us a rapid way to offload distance calculations to BLAS (or equivalent) without looping in our high-level code.

5 How Many Clusters?

A recurring theme in machine learning is model selection, in which we have to decide how big or complicated our representation of the data should be. For clustering algorithms, we make this choice when we fix the number of clusters K. As mentioned previously, large K may be good for feature representations, but smaller K may be more interpretable. Unfortunately, there is no single best way to determine K from the data. Nevertheless there are various useful heuristics in the literature. For a review and comparison, see Gordon (1996). More recent work includes Hamerly and Elkan (2004).

One principled approach is to construct a null hypothesis for the clustering. For example, the idea of the *gap statistic* (Tibshirani et al., 2001) is to compare a dispersion statistic of the clustered fit to the same statistic fit to synthetic data that are known not to have clusters. When we have the right number of clusters, we would expect to have less dispersion than random. Let's imagine that we have run K-Means and we now have a set of responsibilities $\{\boldsymbol{r}_n\}_{n=1}^N$ and cluster centers $\{\boldsymbol{\mu}_k\}_{k=1}^K$. We define the **within-cluster dispersion** to be the sum of squared distances between all pairs in a

[11] Basic Linear Algebra Subprograms: `http://www.netlib.org/blas/`

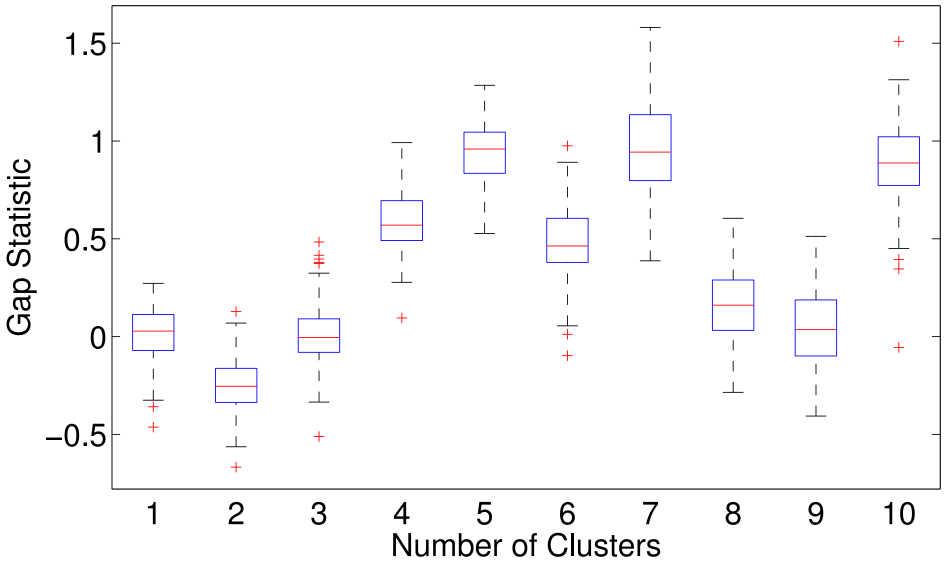


Figure 5: The gap statistic computed for the fruit data. Here, it seems reasonable to choose $K = 5$. This used 100 reference data sets, each from a Gaussian MLE fit for the null model.

given cluster:

$$D_k = \sum_{n=1}^{N} \sum_{n'=1}^{N} r_{n,k}\, r_{n',k}\, ||\boldsymbol{x}_n - \boldsymbol{x}_{n'}||^2. \tag{16}$$

The **dispersion** for a size-K clustering is the normalized sum of the within-cluster dispersion over all of the clusters:

$$W_K = \sum_{k=1}^{K} \frac{D_k}{2N_k} = \sum_{k=1}^{K} \frac{1}{2N_k} \sum_{n=1}^{N} \sum_{n'=1}^{N} r_{n,k}\, r_{n',k}\, ||\boldsymbol{x}_n - \boldsymbol{x}_{n'}||^2 \tag{17}$$

$$N_k = \sum_{n=1}^{N} r_{n,k}. \tag{18}$$

This is basically a measure of “tightness” of the fit normalized by the size of each cluster. It will be smaller when the clustered points group together closely. The gap statistic uses a **null distribution** for the data we have clustered, from which we generate *reference data*. You can think of this as a distribution on the same space, with similar coarse statistics, but in which there aren’t clusters. For example, if our original data were in the unit hypercube $[0, 1]^D$, we might generate reference data uniformly from the cube. If we standardized data on $\mathbb{R}^D$ so that each feature has zero sample mean and unit sample variance, when it would be natural to make the null distribution $\mathcal{N}(0, \mathbf{I}_D)$. To compute the gap statistic, we now generate several sets of reference data, cluster each of them, and compute their dispersions. This gives us an idea (with error bars) as to what the expected dispersion would be based on the coarse properties of the data space, for a given K. We can then

cow	**humpback whale**	**german shepherd**	**mole**
walks	hairless	furry	furry
quadrapedal	toughskin	meatteeth	small
vegetation	big	walks	fast
ground	swims	fast	active
big	strong	quadrapedal	newworld
ox	blue whale	siamese cat	hamster
pig	seal	wolf	rat
sheep	walrus	chihuahua	squirrel
buffalo	dolphin	dalmatian	mouse
horse	killer whale	weasel	skunk

Table 2: This table shows the result of applying K-Medoids to binary features associated with 50 animals, using Hamming distance. Here $K = 4$. The bold animals along the top are the medoids for each cluster. The top five most common features are shown next, followed by five other non-medoid animals from the same cluster.

compute the gap statistic as follows:

$$\text{Gap}_N(K) = \mathbb{E}_{\text{null}}[\log W_K] - \log W_K. \tag{19}$$

Here the first term is the expected log of the dispersion under the null distribution – something we can compute by averaging the log dispersions of our reference data. We subtract from this the dispersion of our actual data. We can then look for the K which maximizes $\text{Gap}_N(K)$. We choose the smallest one that appears to be statistically significant. Figure 5 shows a boxplot of the gap statistic for the fruit data. Here the null distribution was an MLE Gaussian fit to the data. I generated 100 sets of reference data.

6 The K-Medoids Algorithm

As mentioned in the beginning, the distance metric we choose is critical for clustering. When the distances are Euclidean and means correspond to sensible points in $\mathcal{X}$, then K-Means is a natural procedure. However, it may be that we want to use more general distances, or that means are not sensible in our space. In this case, we might want to use a *K-Medoids* procedure. The main difference is that K-Medoids requires the cluster centers to be in the data set. You can think of these special data as "exemplars" for their clusters. K-Medoids is typically more computationally intensive to fit, but can be more interpretable because the clusters are represented by data examples. Algorithm 4 provides a simple implementation of K-Medoids and a couple of examples follow.

Algorithm 4 K-Medoids *Note: written for clarity, not efficiency.*

```
Input: Data vectors $\{\boldsymbol{x}_n\}_{n=1}^N$, number of clusters $K$
for $n \leftarrow 1 \dots N$ do                                  ▷ Initialize all of the responsibilities.
    $\boldsymbol{r}_n \leftarrow [0, 0, \cdots, 0]$                 ▷ Zero out the responsibilities.
    $k' \leftarrow \text{RandomInteger}(1, K)$         ▷ Make one of them randomly one to initialize.
    $r_{nk'} = 1$
end for
repeat
    for $k \leftarrow 1 \dots K$ do                                        ▷ Loop over the clusters.
        for $n \leftarrow 1 \dots N$ do                                        ▷ Loop over data.
            if $r_{n,k} = 1$ then
                $J_n \leftarrow \sum_{n'=1}^N r_{n',k} ||\boldsymbol{x}_n - \boldsymbol{x}_{n'}||$     ▷ Sum distances to this datum.
            else
                $J_n \leftarrow \infty$                      ▷ Infinite cost for data not in this cluster.
            end if
        end for
        $n^\star \leftarrow \arg\min_n J_n$         ▷ Pick the one that minimizes the sum of distances.
        $\boldsymbol{\mu}_k \leftarrow \boldsymbol{x}_{n^\star}$                ▷ Make the minimizing one the cluster center.
    end for
    for $n \leftarrow 1 \dots N$ do                                        ▷ Loop over the data.
        $\boldsymbol{r}_n \leftarrow [0, 0, \cdots, 0]$                 ▷ Zero out the responsibilities.
        $k' \leftarrow \arg\min_k ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2$               ▷ Find the closest medoid.
        $r_{nk'} = 1$
    end for
until none of the $\boldsymbol{r}_n$ change
Return assignments $\{\boldsymbol{r}_n\}_{n=1}^N$ for each datum, and cluster medoids $\{\boldsymbol{\mu}_k\}_{k=1}^K$.
```

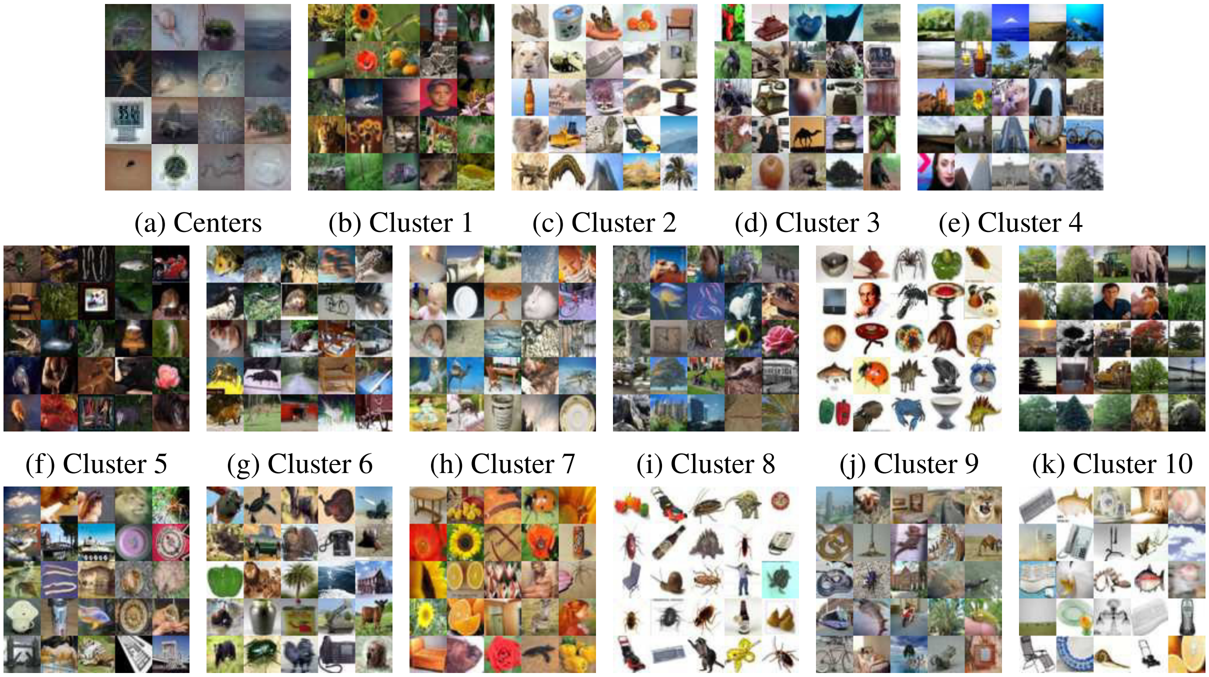
6.1 Example: Binary Animal Properties

Table 2 shows the result of applying K-Medoids clustering to binary feature data for different types of animals.[12] The animals are things like “weasel” and “horse”, while the binary features are properties such as “swims” or “furry”. There are 50 animals and 85 features. I used $K = 4$ and Hamming distance. I initialized with K-Means++. The table has four columns with the medoid at the top, the most common five features within the group, and five other animals in the cluster, excluding the medoid.

6.2 Example: Image Data

Figure 6 shows the result of K-Medoids applied to 50000 32×32 color images. These are the same CIFAR-100 data as used in the K-Means example. The only difference here is that I used

[12] `http://www.psy.cmu.edu/~ckemp/code/irm.html`



(a) Centers (b) Cluster 1 (c) Cluster 2 (d) Cluster 3 (e) Cluster 4

(f) Cluster 5 (g) Cluster 6 (h) Cluster 7 (i) Cluster 8 (j) Cluster 9 (k) Cluster 10

(l) Cluster 11 (m) Cluster 12 (n) Cluster 13 (o) Cluster 14 (p) Cluster 15 (q) Cluster 16

Figure 6: This is the result of K-Medoids clustering applied to the CIFAR-100 image data. (a) The 16 cluster medoids. (b-q) 25 data examples are shown for each of the 16 clusters.

K-Medoids clustering instead. Note that the medoids tend to have low spatial frequency.

7 Advanced Topics

These topics are outside the scope of this class, but are good things to look into next if you find clustering to be an interesting topic.

Spectral Clustering Clusters in data are often more complicated than simple isotropic blobs. Our human visual system likes to group things in more complicated ways. Spectral clustering (see, e.g., Ng et al. (2002); Von Luxburg (2007)) constructs a graph over the data first and then performs operations on that graph using the Laplacian. The idea is that data which are close together tend to be in the same group, even if they are not close to some single prototype.

Affinity Propagation One powerful way to think about clustering, which we'll see later in the semester, is to frame the groups in terms of latent variables in a probabilistic model. Inference in many probabilistic models can be performed efficiently using "message passing" algorithms which take advantage of an underlying graph structure. The algorithm of *affinity propagation* (Frey and Dueck, 2007) is a nice way to perform K-Medoids clustering efficiently with such a message passing procedure.

Biclustering The clustering algorithms we've looked at here have operated on the data instances. That is, we've thought about finding partitions of the rows of an $N \times D$ matrix. Many data are not well represented by "instances" and "features", but are interactions between items. For example, we could imagine our data to be a matrix of outcomes between sports teams, or protein-protein interaction data. Biclustering (Hartigan, 1972) is an algorithm for simultaneously grouping both rows and columns of a matrix, in effect discovering blocks. Variants of this technique have become immensely important to biological data analysis.

References

Adam Coates, Andrew Y. Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, pages 215–223, 2011. URL `http://www.stanford.edu/~acoates/papers/coatesleeng_aistats_2011.pdf`.

Stuart Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28 (2):129–137, 1982. URL `http://www.nt.tuwien.ac.at/fileadmin/courses/389075/Least_Squares_Quantization_in_PCM.pdf`.

Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, 2009. URL `http://link.springer.com/article/10.1007%2Fs10994-009-5103-0`.

David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007. URL `http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf`.

Allan D. Gordon. Null models in cluster validation. In *From data to knowledge*, pages 32–44. Springer, 1996.

Greg Hamerly and Charles Elkan. Learning the k in k-means. *Advances in neural information processing systems*, 16:281, 2004.

Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001. URL `http://www.stanford.edu/~hastie/Papers/gap.pdf`.

Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002. URL `http://machinelearning.wustl.edu/mlpapers/paper_files/nips02-AA35.pdf`.

Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007. URL `http://web.mit.edu/~wingated/www/introductions/tutorial_on_spectral_clustering.pdf`.

Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

John A Hartigan. Direct clustering of a data matrix. *Journal of the american statistical association*, 67(337):123–129, 1972.

Changelog

- 2 November 2018 – Initial revision from old CS181 notes.