Unsupervised Learning

Unsupervised machine learning is the machine learning task of inferring a function to describe hidden structure from "unlabeled" data:

- a classification or categorization is not included in the observations

Since the examples given to the learner are unlabeled, there is no easy evaluation of the accuracy of the structure that is output by the relevant algorithm—which is one way of distinguishing unsupervised learning from supervised learning.

Unsupervised Learning

Here we will discuss a class of unsupervised machine learning models: *clustering algorithms.*

Clustering algorithms seek to learn, from the properties of the data, an optimal *division* or discrete labeling of *groups of points*.

Perhaps the most popular clustering algorithm is the *k-means* algorithm.

The k-Means Algorithm

The *k*-means algorithm searches for a *pre-determined* number of clusters within an unlabeled multidimensional dataset.

It accomplishes this using a simple conception of what the optimal clustering looks like:

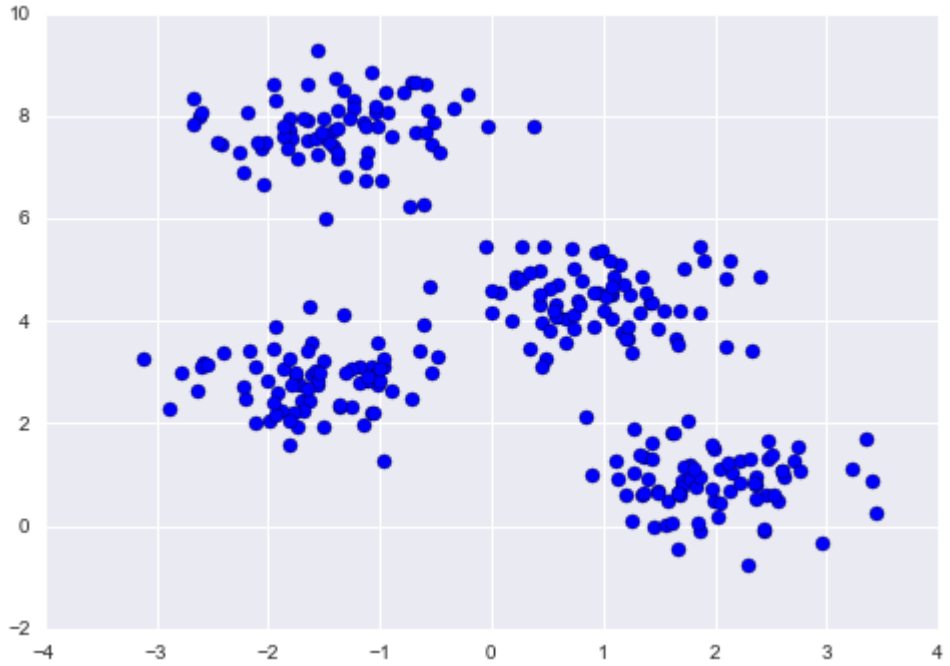
- The "cluster center" is the *arithmetic mean* of all the points belonging to the cluster.
- Each point is closer to its own cluster center than to other cluster centers.

Those two assumptions are the basis of the *k*-means model.

https://en.wikipedia.org/wiki/K-means_clustering

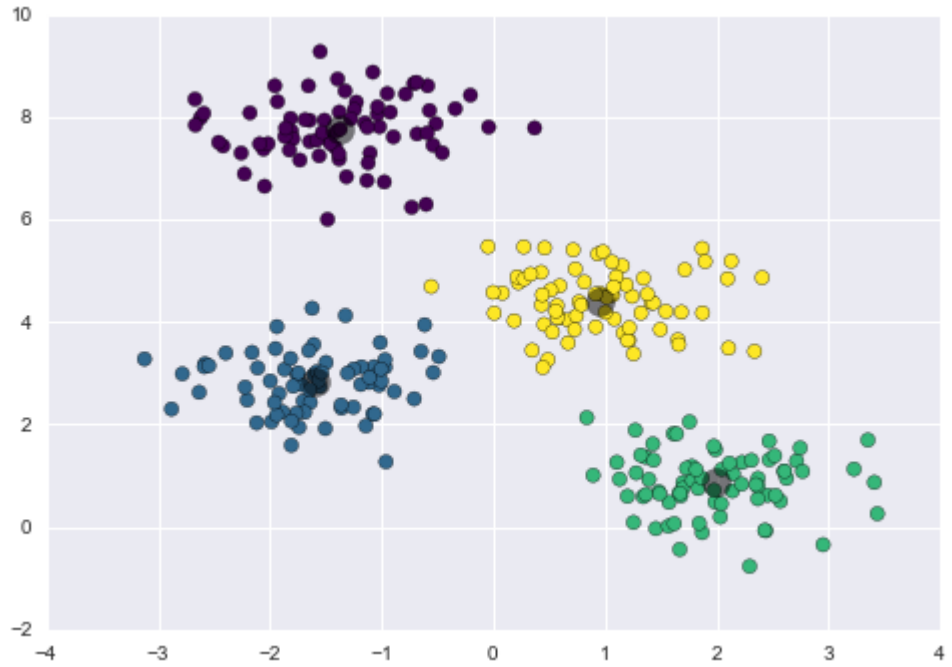
The k-Means Algorithm

```
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=300, centers=4,
                       cluster_std=0.60, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50);
```



The k-Means Algorithm

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)

plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
```



k-Means Algorithm: Expectation Maximization

The expectation–maximization approach here consists of the following procedure:

1. Guess some cluster centers
2. Repeat until converged
    1. *E-Step*: assign points to the nearest cluster center
    2. *M-Step*: set the cluster centers to the mean

Here the "E-step" or "Expectation step" is so-named because it involves updating our expectation of which cluster each point belongs to - in this context expectation is just a fancy word of mean/average.

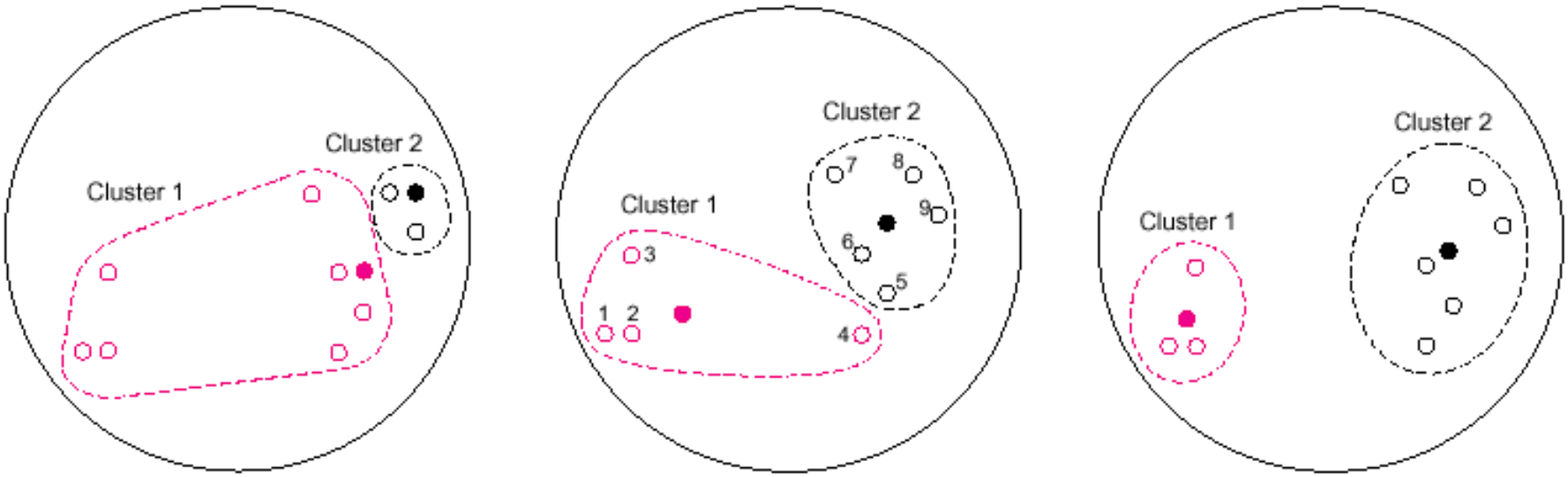
The "M-step" or "Maximization step" is so-named because it involves maximizing the mean of the data in each cluster.

The k-Means Algorithm

(a) Setup:
Reference point 1 (filled red circle) and reference point 2 (filled black circle) are chosen arbitrarily. All data points (open circles) are then partitioned into two clusters: each data point is assigned to cluster 1 or cluster 2, depending on whether the data point is closer to reference point 1 or 2, respectively.

(b) Results of first iteration:
Next each reference point is moved to the centroid of its cluster. Then each data point is considered in the sequence shown. If the reference point closest to the data point belongs to the other cluster, the data point is reassigned to that other cluster, and both cluster centroids are recomputed.

(c) Results of second iteration:
During the second iteration, the process in Figure 3(b) is performed again for every data point. The partition shown above is stable; it will not change for any further iteration.



Iterations of the k-means algorithm with $k = 2$.

Visualizing the k-Means Algorithm

http://stanford.edu/class/ee103/visualizations/kmeans/kmeans.html

The k-Means Algorithm

- Pros:
  - Fast convergence
  - Conceptually simple
- Cons:
  - Very sensitive to the choice of numbers of clusters *k*
  - Only considers convex cluster boundaries

k-Means: selection of k

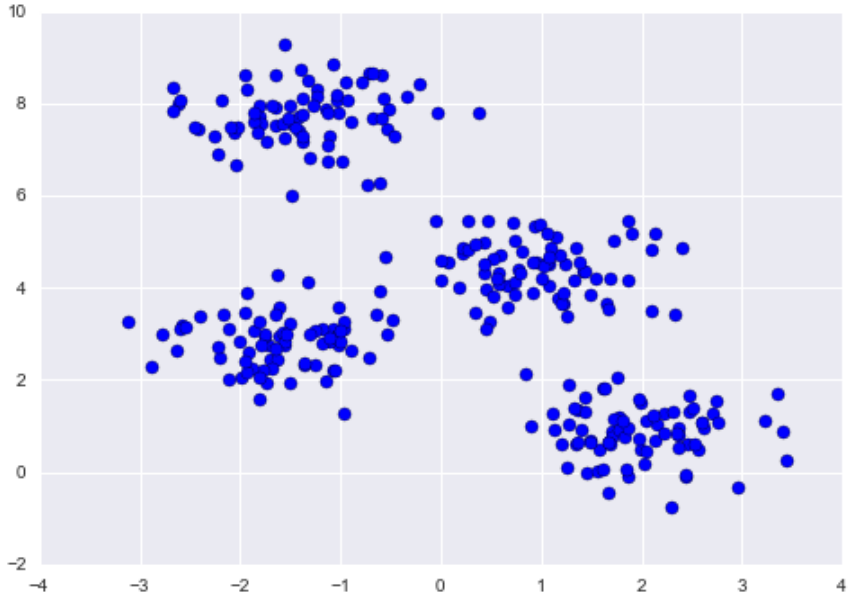
The selection of k can have a serious effect on the quality of your clusters.

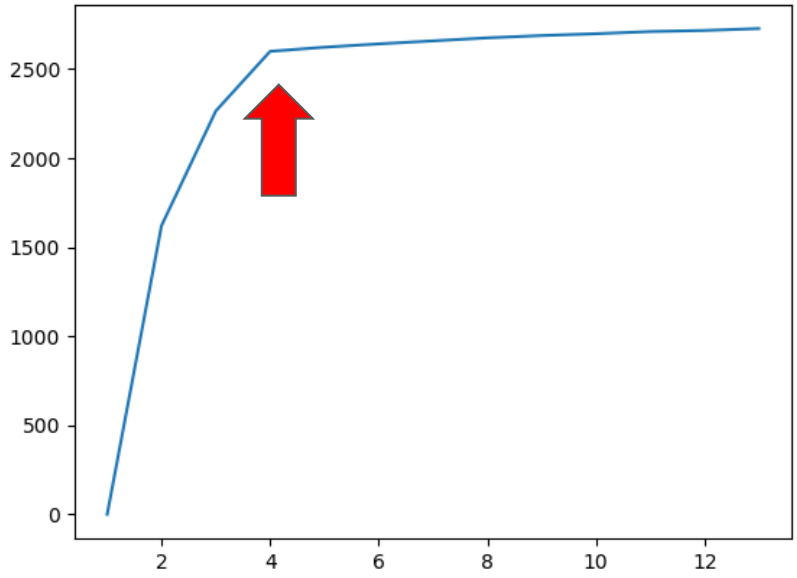
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

labels = KMeans(6, random_state=0).fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
plt.show()
```

k-Means: The Elbow method

The Elbow method is a method of interpretation and validation of consistency within cluster analysis designed to help finding the appropriate number of clusters in a dataset.

Looking for the ‘elbow’ in a plot





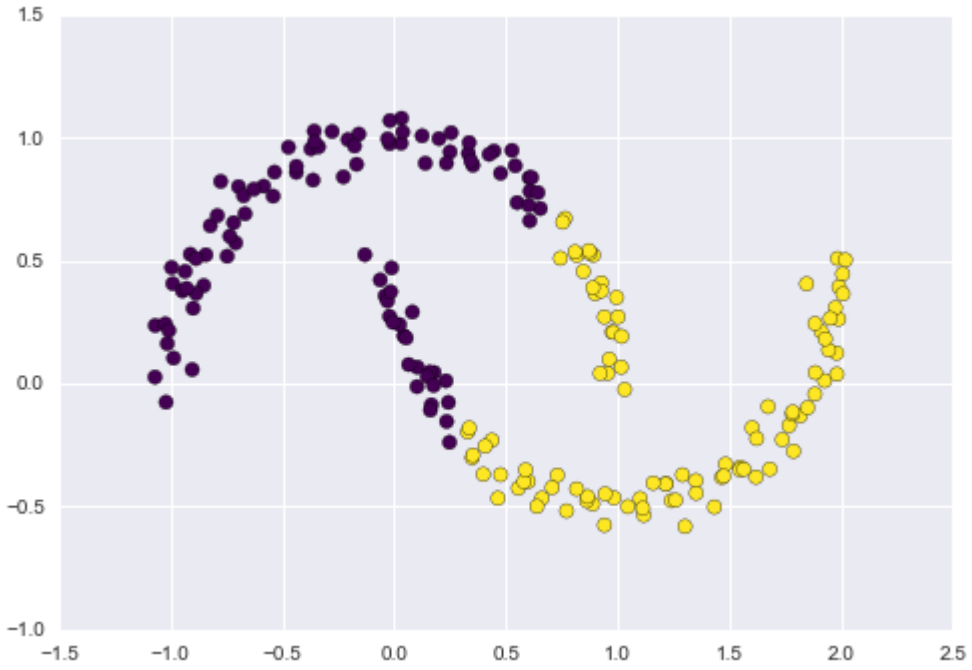
k-Means: The Elbow method

```
def elbow(df, n):
    import matplotlib.pyplot as plt
    from sklearn.cluster import KMeans
    import numpy as np
    from scipy.spatial.distance import cdist, pdist
    # kmeans models for each k
    kMeansVar = [KMeans(n_clusters=k).fit(df.values) for k in range(1, n)]
    # get the centroids of the models
    centroids = [X.cluster_centers_ for X in kMeansVar]
    # find the distances of the values to the centroids
    k_euclid = [cdist(df.values, cent) for cent in centroids]
    # find the distance of each point to its cluster center
    dist = [np.min(ke, axis=1) for ke in k_euclid]
    # total within cluster sum of squares
    wcss = [sum(d**2) for d in dist]
    # total sum of squares
    tss = sum(pdist(df.values)**2)/df.values.shape[0]
    # between clusters sum of squares
    bss = tss - wcss
    plt.plot(list(range(1,n)),bss)
    plt.show()
```

Demo elbow

k-Means: non-linear boundaries

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_moons

X, y = make_moons(200, noise=.05, random_state=0)
labels = KMeans(2, random_state=0).fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis');
```

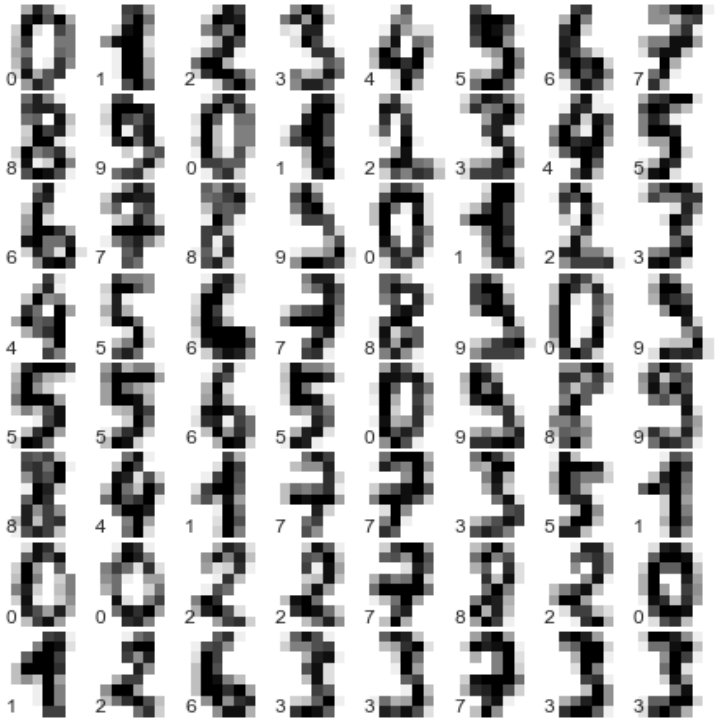


Example: Clustering Digits

The digits data set consists of 1,797 samples each consisting of an 8 × 8 grid of pixels (64 features) representing a handwritten digits between 0 and 9.

Here are a few samples from this data set.

```
from sklearn.datasets import load_digits
digits = load_digits()
```



Question: can we use k-means to cluster in this data set and retrieve the digits?

Example: Clustering Digits

The idea is to set up 10 clusters and then each cluster should contain all the rows representing one of the digits.

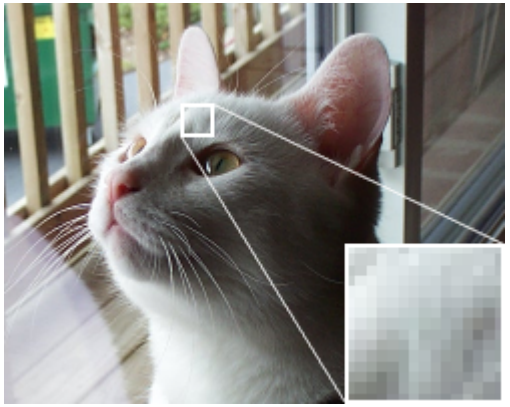
The result is 10 clusters in 64 dimensions. Notice that the cluster centers themselves are 64-dimensional points, and can themselves be interpreted as the "typical" digit within the cluster.

```
kmeans = KMeans(n_clusters=10, random_state=0)
kmeans.fit(digits.data)

fig, ax = plt.subplots(2, 5, figsize=(8, 3))
centers = kmeans.cluster_centers_.reshape(10, 8, 8)
for axi, center in zip(ax.flat, centers):
    axi.set(xticks=[], yticks=[])
    axi.imshow(center, interpolation='nearest', cmap=plt.cm.binary)
```

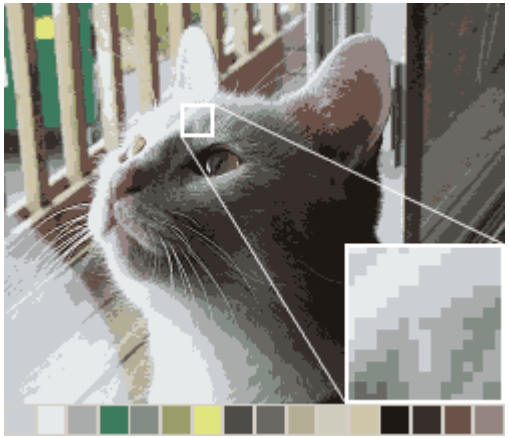
We see that even without the labels, KMeans is able to find clusters whose centers are recognizable digits, with perhaps the exception of 1 and 8.

Example: Color Quantization

In computer graphics, color quantization or color image quantization is a process that reduces the number of distinct colors used in an image, usually with the intention that the new image should be as visually similar as possible to the original image.



An example image in 24-bit RGB color



The same image reduced to a palette of 16 colors specifically chosen to best represent the image; the selected palette is shown by the squares above

Source: Wikipedia

Example: Color Quantization

Most standard techniques treat color quantization as a problem of clustering points in three-dimensional space, where the points represent colors found in the original image and the three axes represent the three color channels.

Almost any three-dimensional clustering algorithm can be applied to color quantization, including k-Means.

After the clusters are located, typically the points in each cluster are averaged to obtain the representative color that all colors in that cluster are mapped to.

The three color channels are usually red, green, and blue.

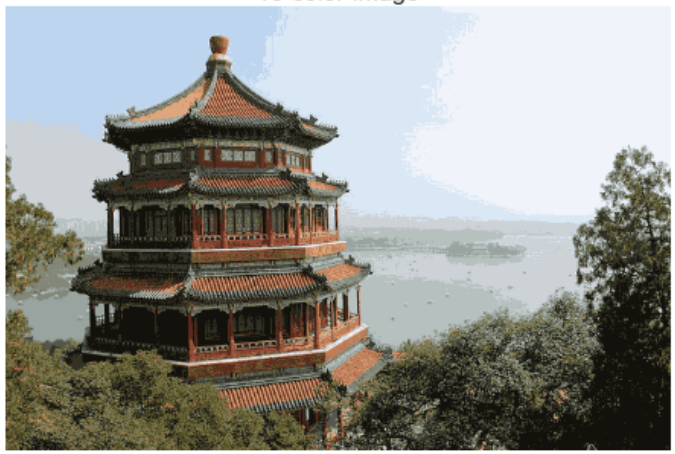
Example: Color Quantization

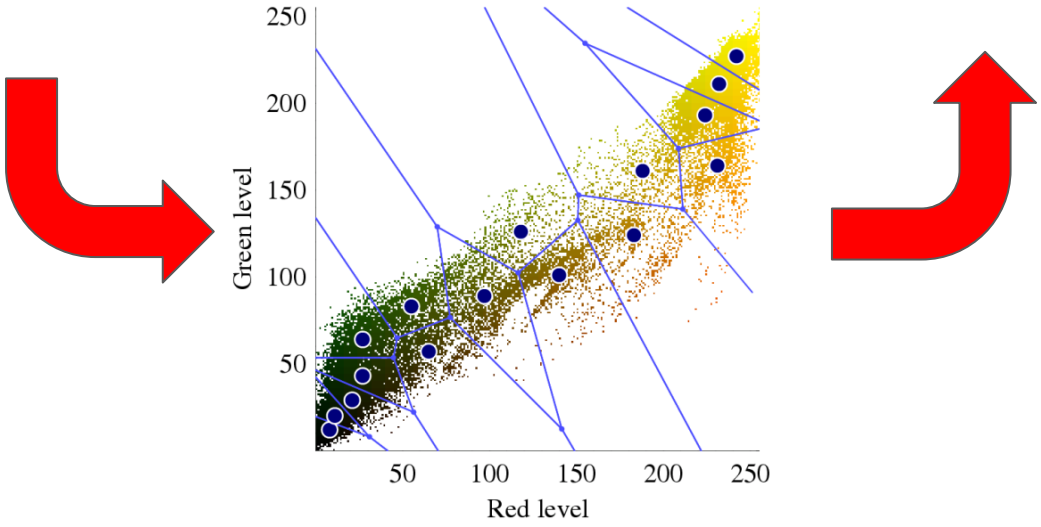
```
>>> data = china / 255.0 # use 0...1 scale
>>> data = data.reshape(427 * 640, 3)
>>> data.shape
(273280, 3)
```

Let's apply k-Means as a color quantizer.

Original Image

16-color Image



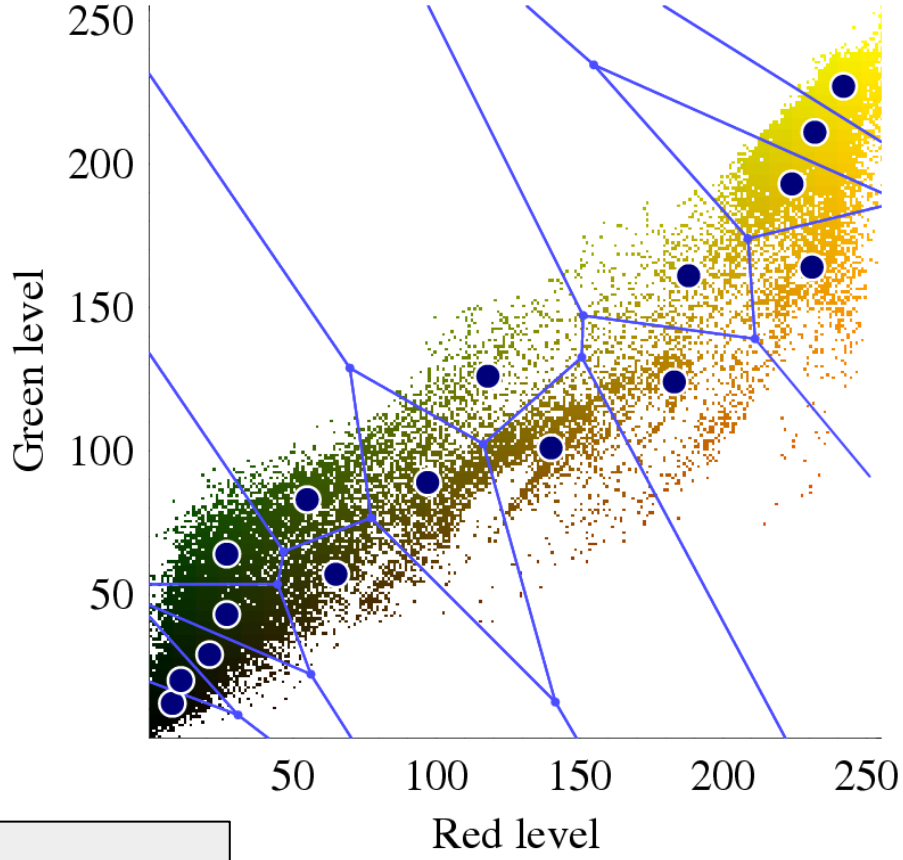


Example: Color Quantization

k-Means cluster partition space into a *Voronoi diagram*.

All the points in a particular Voronoi diagram partition will be coded with the average color for that partition.

Compressing millions of colors into 16 colors.




```
from sklearn.cluster import MiniBatchKMeans
kmeans = MiniBatchKMeans(16)
kmeans.fit(data)
new_colors = kmeans.cluster_centers_[kmeans.predict(data)]
```

Thursday 4/19 - Quantum Computing Demo

- We will have a guest speaker on Thursday 4/19
- Lecture and lab on quantum computing
- Please sign up for an account before the lecture:
  - http://research.ibm.com/ibm-q/quantum-card-test/
  - Bottom of page under ‘Experiment’

Final Project Proposals

Final project proposals are due on Monday 4/23.

As in the midterm a final project could be a (team) app (e.g. Python script that implements some non-trivial functionality with/without visualization) or it can be an individual analysis project.

However, whether app or analysis your final project has to include some modeling aspect – regression or clustering