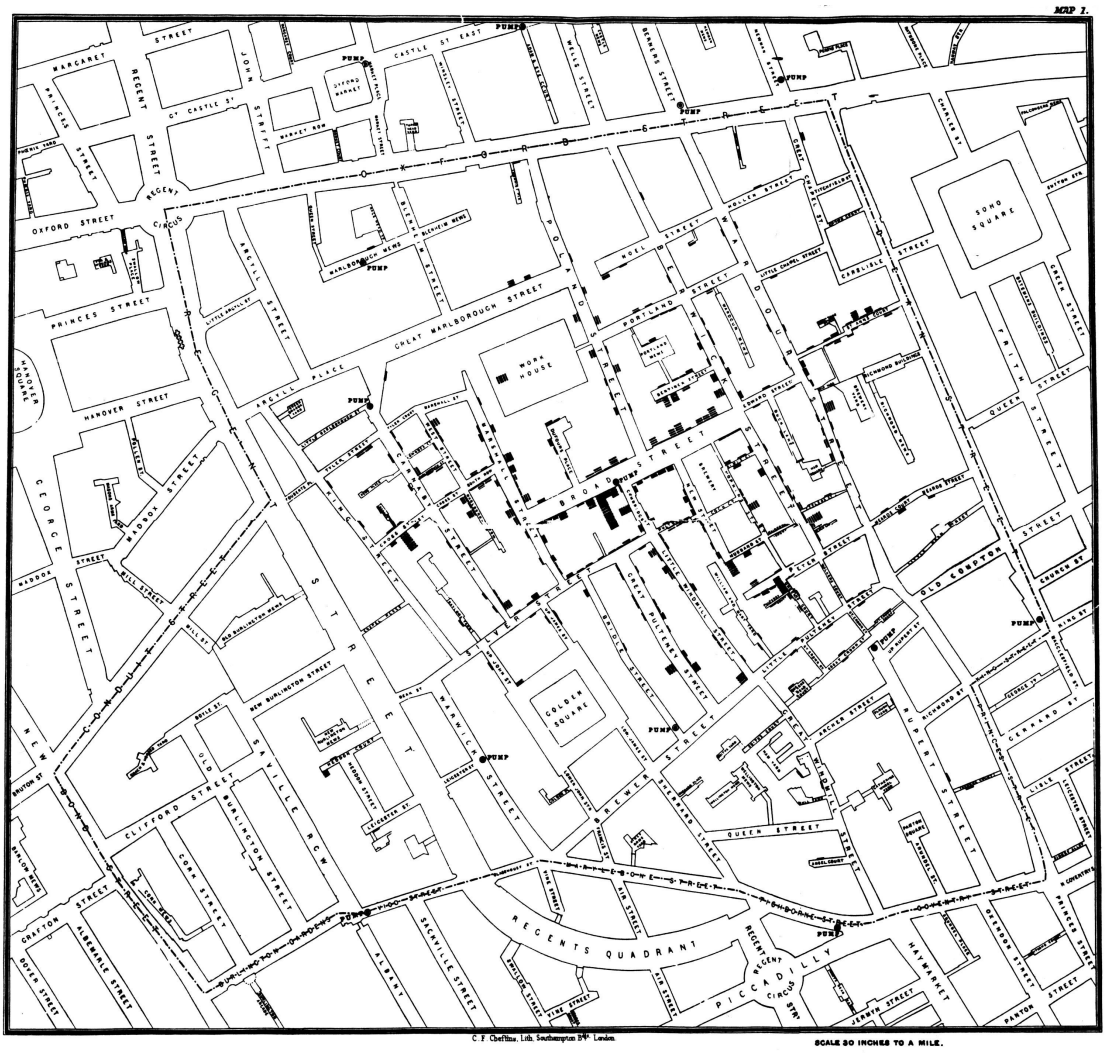
06A-Clustering-Overview

September 21, 2017

1 Clustering

1854: Cholera outbreak in Soho, London.
Common wisdom at the time was that disease spread by breathing "foul air" (miasma).
In 10 days, 500 people in the area died.
John Snow: local physician.
Sewer system had not yet reached Soho. Most homes had cesspits under the floor.



By John Snow - Published by C.F. Cheffins, Lith, Southhampton Buildings, London, England, 1854 in Snow, John. On the Mode of Communication of Cholera, 2nd Ed, John Churchill, New Burlington Street, London, England, 1855.

(This image was originally from en.wikipedia; description page is/was here. Image copied fromhttp://matrix.msu.edu/~johnsnow/images/online_companion/chapter_images/fig12-5.jpg), Public Domain, https://commons.wikimedia.org/w/index.php?curid=2278605

1.1 Clustering

Clustering is a very important way of discovering **structure** in data.

It is so important because it is **common** for data to show clusters.

- Locations where millionaires live
- The number of hours people work each week
- Demographics ("soccer moms", "bored retirees", "unemployed millenials", etc)

However, these categories or "labels" have been assigned **after** the fact.

When we do clustering, we first find clusters, then **interpret** or "assign labels."

That is, clustering represents the first example we will see of **unsupervised** learning.

Supervised methods: Data items have labels, and we want to learn a function that correctly assigns labels to new data items.

Unsupervised methods: Data items do not have labels, and we want to learn a function that extracts important patterns from the data.

Feature scaling

When constructing or selecting a distance metric, one needs to think carefully about the scale of the features being used.

For example, consider the case where we are clustering people based on their age, income, and gender.

We might use age in years, income in dollars, and assign gender to the values $\{0, 1\}$.

Thus, the following records:

- Joe Smith, age 27, income USD 75,000, male
- Eve Jones, age 45, income USD 42,000, female

Would be encoded in feature space as:

$$\begin{bmatrix} 27 \\ 75000 \\ 0 \end{bmatrix}, \begin{bmatrix} 45 \\ 42000 \\ 1 \end{bmatrix}$$

What would happen if we used Euclidean distance as our dissimilarity metric in this feature space?

Clearly, the influence of income would dominate the other two features. For example, a difference of gender is about as significant as a difference of one dollar of yearly income.

We are unlikely to expose gender-based differences if we cluster using this representation.

The most common way to handle this is **feature scaling.**

The basic idea is to rescale each feature separately, so that its range of values is about the same as all other features.

For example, one may choose to:

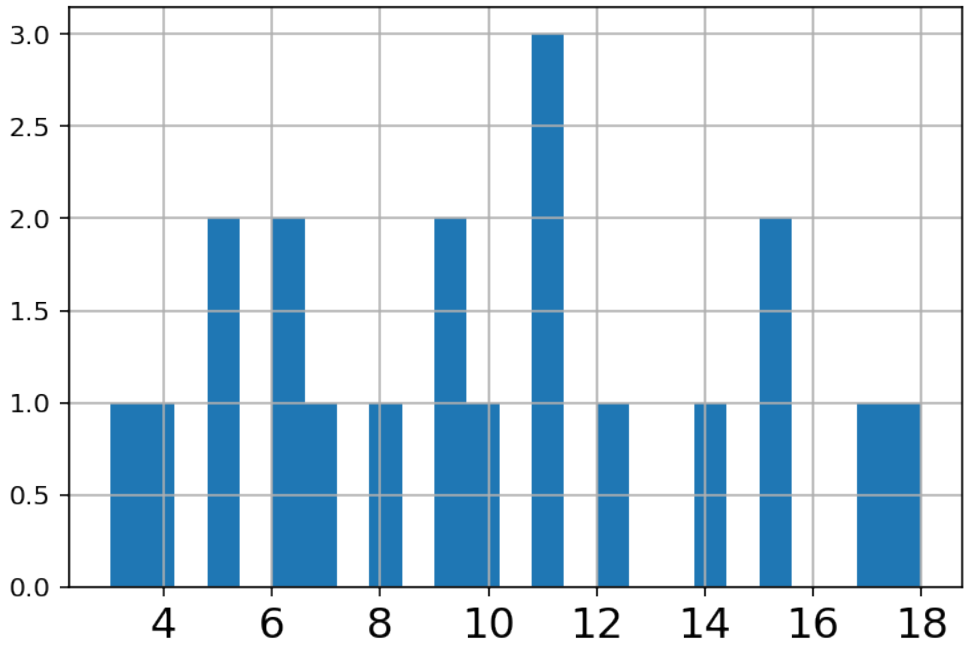
- shift each feature independently by subtracting the mean over all observed values
  - This means that each feature is now centered on zero
- then rescale each feature so that the standard deviation overall observed values is 1.
  - This means that the feature will have about the same range of values as all the others.

For example, let's work with Bortkiewicz's famous horse-kick data:

In [62]:
```
import pandas as pd
df = pd.read_table('data/HorseKicks.txt',index_col='Year',dtype='float')
counts = df.sum(axis=1)
counts
```

Out[62]:
```
Year
1875.0     3.0
1876.0     5.0
1877.0     7.0
1878.0     9.0
1879.0    10.0
1880.0    18.0
1881.0     6.0
1882.0    14.0
1883.0    11.0
1884.0     9.0
1885.0     5.0
1886.0    11.0
1887.0    15.0
1888.0     6.0
1889.0    11.0
1890.0    17.0
1891.0    12.0
1892.0    15.0
1893.0     8.0
1894.0     4.0
dtype: float64
```

In [61]:
```
_ = counts.hist(bins=25,xlabelsize=16)
```



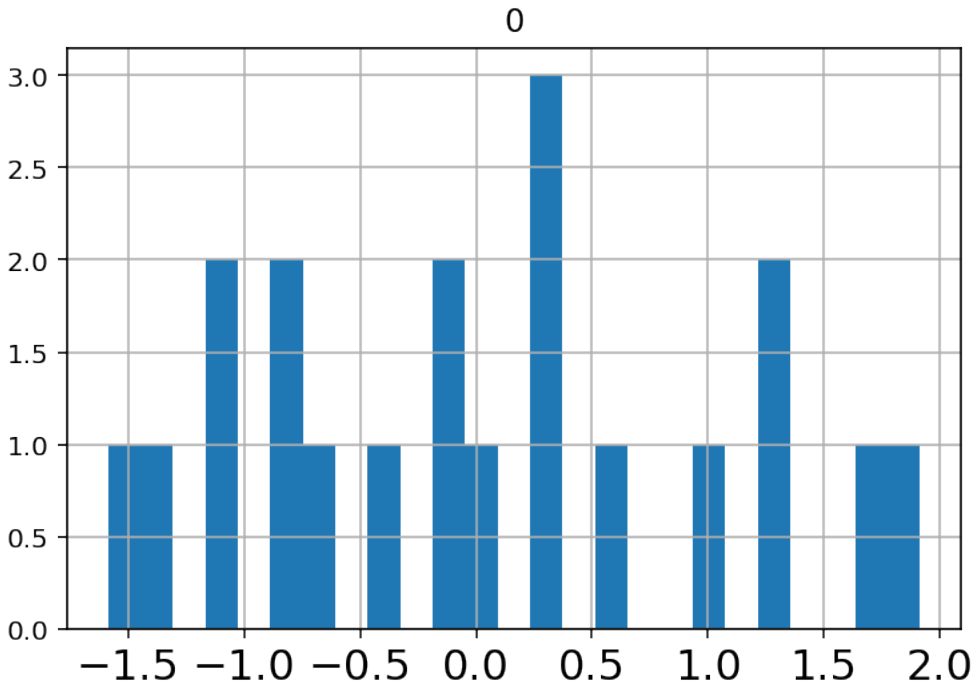
```
In [58]: counts.mean()
```

```
Out[58]: 9.8
```

To standardize to zero mean and unit standard deviation, we can use tools from the `scikit-learn` library.

(We will discuss scikit-learn more in upcoming lectures.)

```
In [49]: from sklearn import preprocessing
         counts_scaled = pd.DataFrame(preprocessing.scale(counts))
         _ = counts_scaled.hist(bins=25,xlabelsize=16)
```



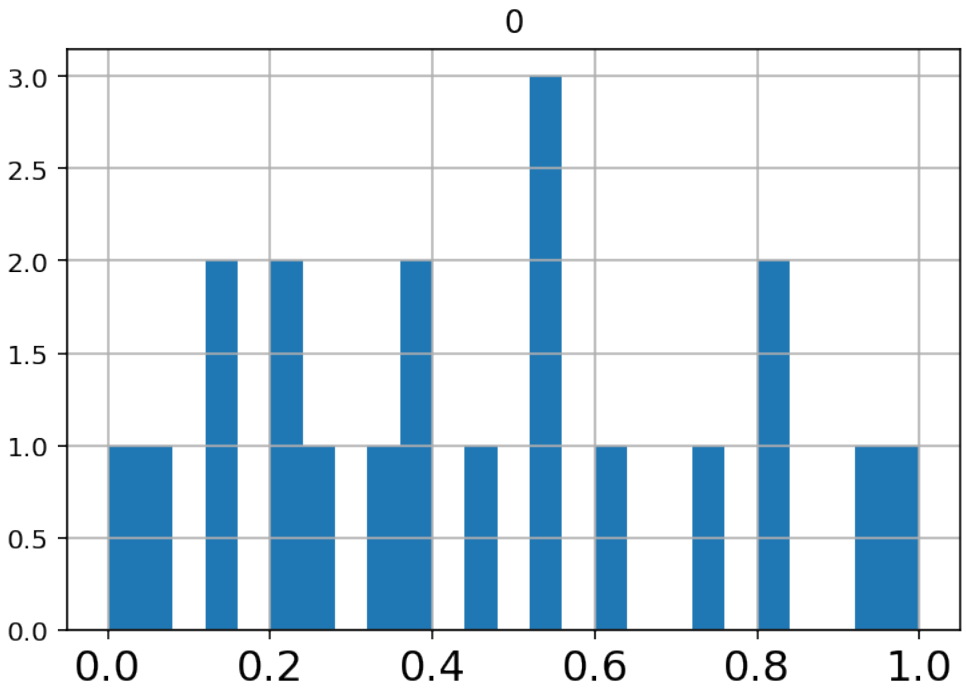

```
In [60]: counts_scaled.mean().values
```

```
Out[60]: array([ -1.77635684e-16])
```

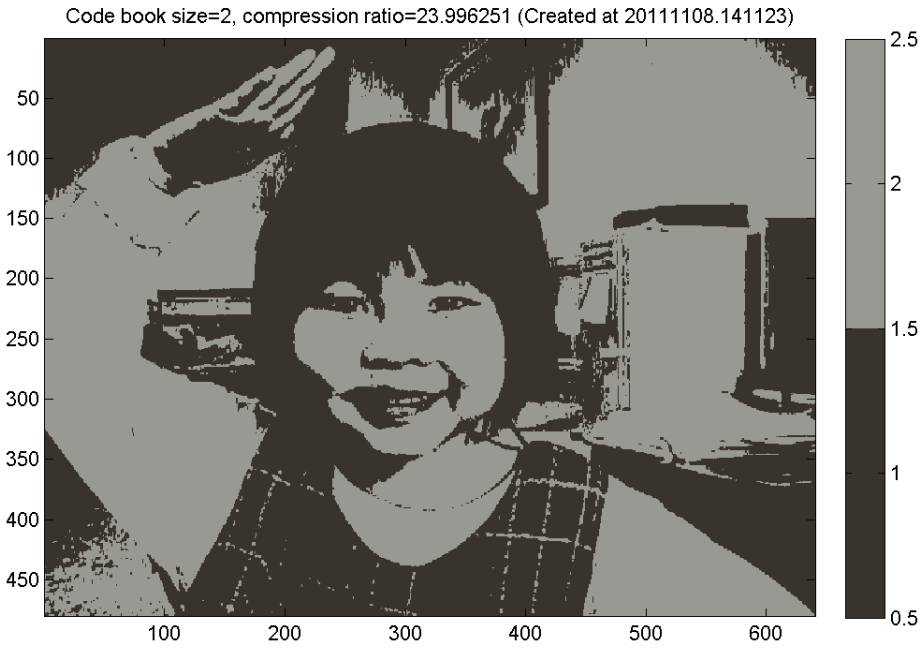
Notice that values that used to be zero have now become negative.

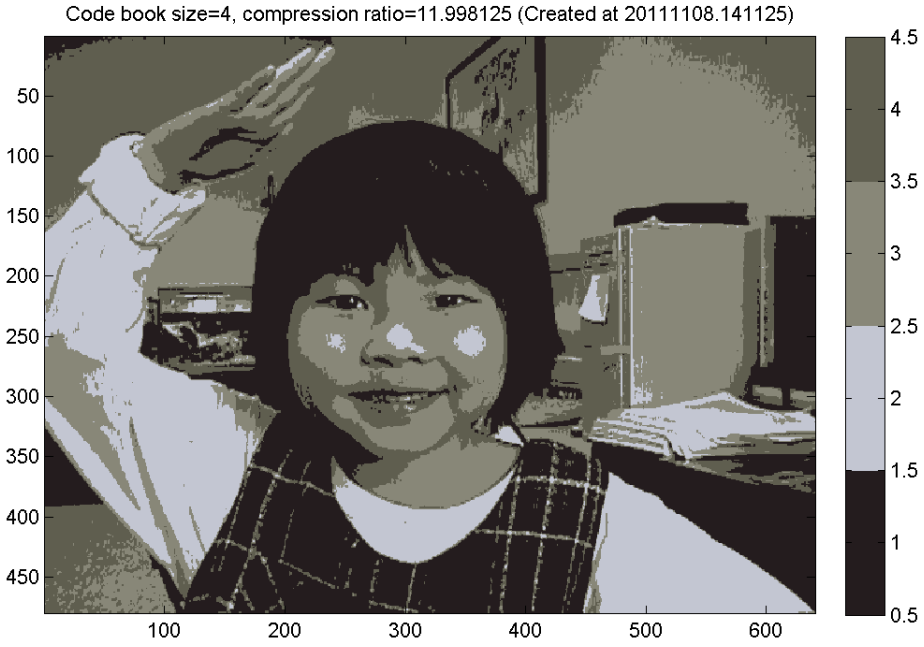
In some situations it may not be sensible to change zeros into something else. It may make more sense to map all values into a fixed range, for example $[0, 1]$.

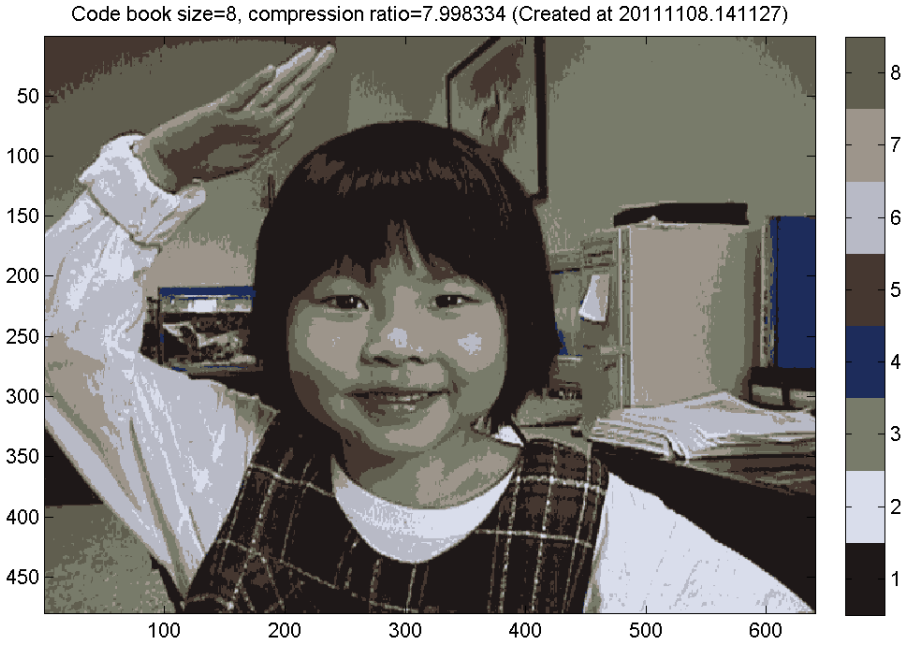
```
In [57]: min_max_scaler = preprocessing.MinMaxScaler()
         counts_minmax = min_max_scaler.fit_transform(counts.values.reshape(-1,1))
         counts_minmax = pd.DataFrame(counts_minmax)
         _ = counts_minmax.hist(bins=25,xlabelsize=16)
```



Example Output from k-means


Code book size=2, compression ratio=23.996251 (Created at 20111108.141123)
50
100
150
200
250
300
350
400
450
100
200
300
400
500
600
2.5
2
1.5
1
0.5


Code book size=4, compression ratio=11.998125 (Created at 20111108.141125)
50
100
150
200
250
300
350
400
450
100
200
300
400
500
600
4.5
4
3.5
3
2.5
2
1.5
1
0.5


Code book size=8, compression ratio=7.998334 (Created at 20111108.141127)
50
100
150
200
250
300
350
400
450
100
200
300
400
500
600
8
7
6
5
4
3
2
1


Code book size=16, compression ratio=5.998126 (Created at 20111108.141138)
50
100
150
200
250
300
350
400
450
100
200
300
400
500
600
16
14
12
10
8
6
4
2