K-Means II

M. R. Hasan

CSCE 411/811

Data Modeling for Systems Development

Readings

- Bishop: 9.1, 9.1.1
- Murphy: 11.4.2.5, 11.4.2.6, 11.4.2.7
- Geron: 9

What We Will Cover

- K-Means: Practical Issues
- Resolving Practical Issues (K++ Means)
- Applications

K-Means Algorithm: Practical Issues

K-Means: Practical Issues

- There are **three practical issues** related to the convergence of the k-Means algorithm.
  - Varity of Distance Measures
  - Data Standardization
  - Convergence to the Global Minimum

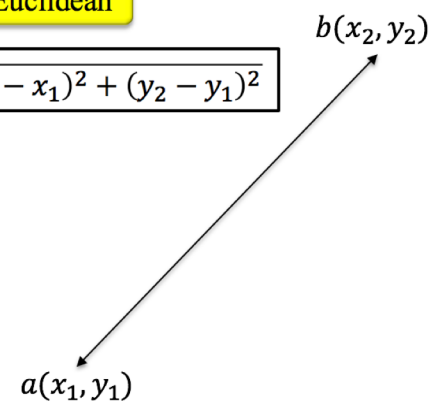
K-Means: Practical Issues

- Varity of Distance Measures:
- The K-Means algorithm ***depends*** on how we define **distance** between samples.
- **Euclidean distance** is the most commonly used metric.
- However, as we will see later that for **high-dimensional data** Euclidean distance may **not be a good measure** of "similarity".
- We may have to use other distance measures such as **Manhattan distance**.

Euclidean vs Manhattan Distance

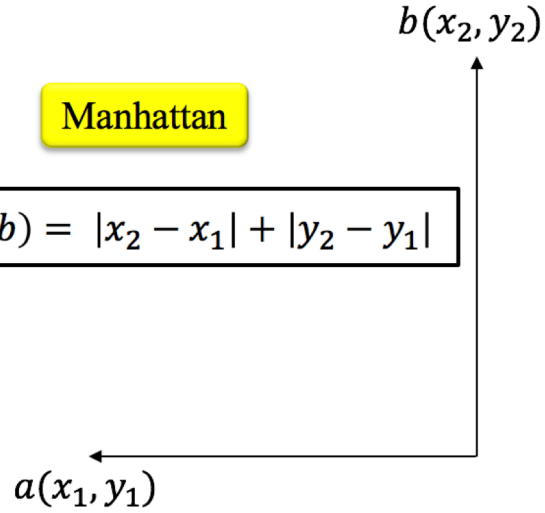
Euclidean

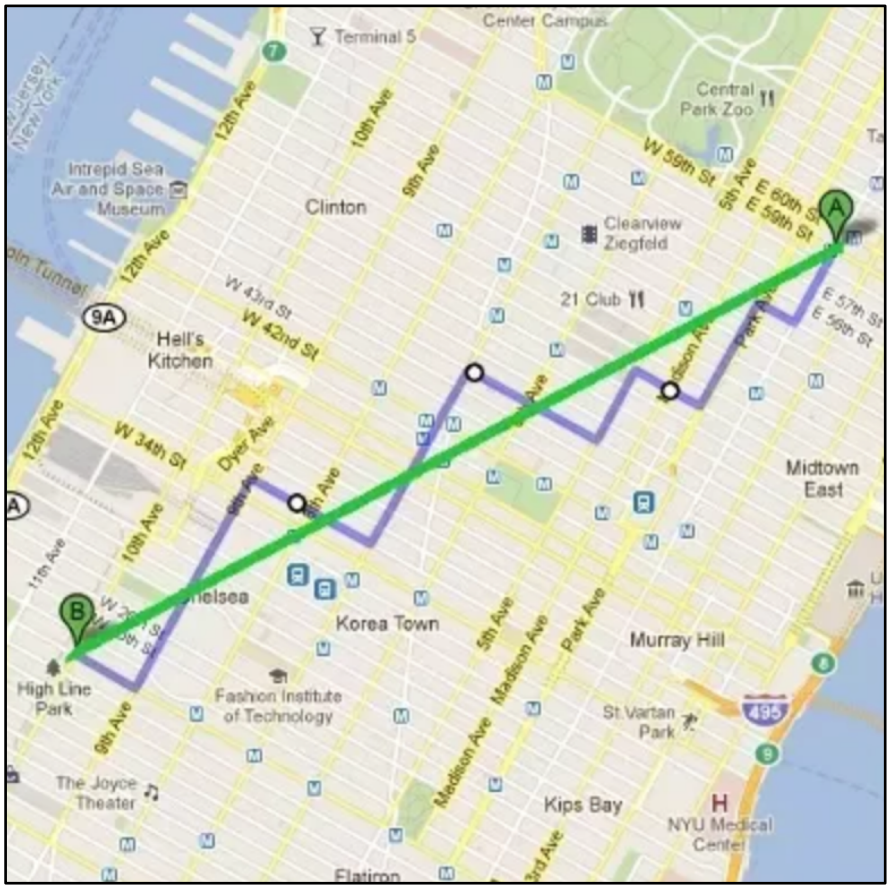
$$d(a,b) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



Manhattan

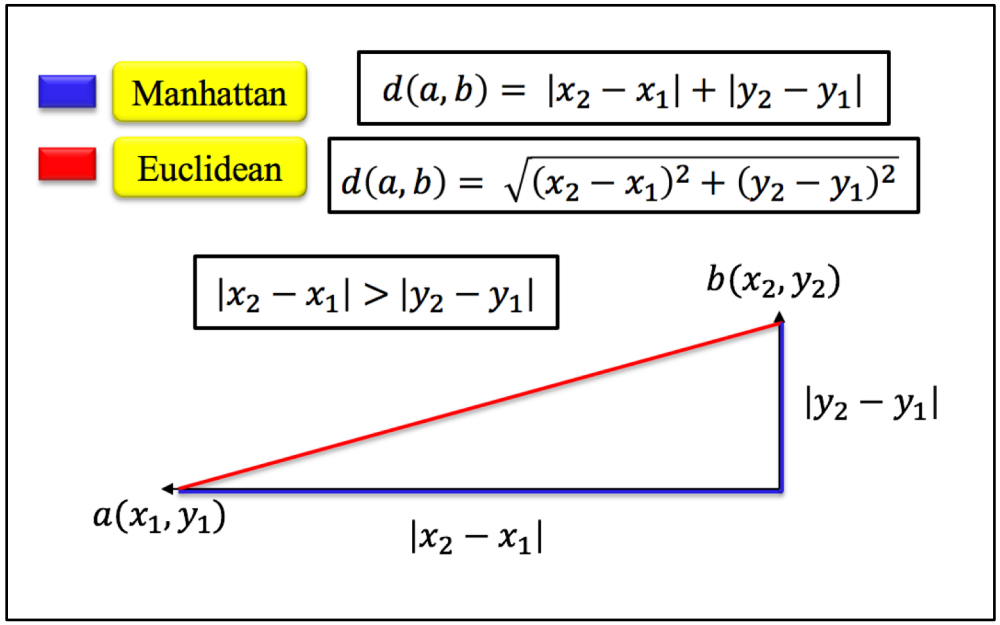
$$d(a,b) = |x_2 - x_1| + |y_2 - y_1|$$





Which distance metric should we use?

Euclidean vs Manhattan Distance



In **high-dimensional** data, the Euclidean distance could ***ignore most of the dimensions.***

Thus, in high-dimensional data, the **Manhattan distance** metric will perform better.

Which distance metric should we use?

It **depends on the dimension** (no. of the features) of the data

In the example, the vectors a and b has the **highest difference** along the x dimension (axis)

Euclidean: the distance between a and b is **dominated by the difference along x dimension**

Manhattan: **both x and y dimensions contribute equally** to measure the distance between a and b

K-Means: Practical Issues

- We ***generalize*** the distance metric for **d-dimensional features** by presenting the **Minkowski distance** metric or **L_P norm**.

$$L_p(\vec{x}, \vec{z}) := d(\vec{x}, \vec{z}) = \left[\sum_{i=1}^{d} |x_i - z_i|^p\right]^{1/p}$$

p = 2: Euclidean Distance

$$L_2(\vec{x}, \vec{z}) := d(\vec{x}, \vec{z}) = \left[\sum_{i=1}^{d} |x_i - z_i|^2\right]^{1/2}$$

p = 1: Manhattan Distance

$$L_1(\vec{x}, \vec{z}) := d(\vec{x}, \vec{z}) = \sum_{i=1}^{d} |x_i - z_i|$$

K-Means: Practical Issues

- Varity of Distance Measures:
- For more discussion on the distance metric see the slides "***Clustering-KMeans-Distance Metric***"

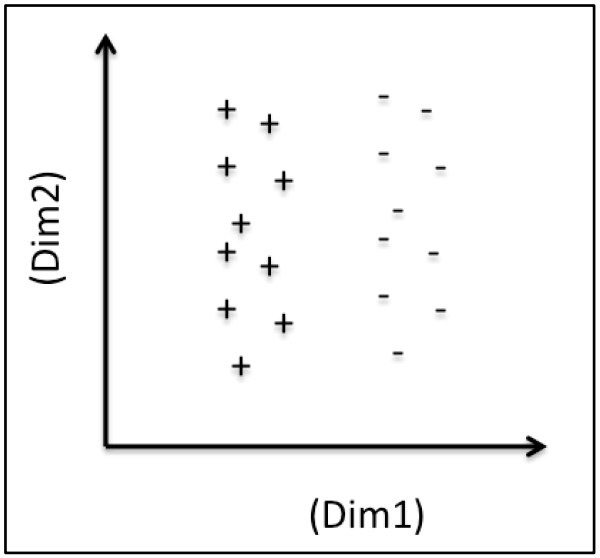
K-Means: Practical Issues

- Data Standardization:
- It is very important that **data is standardized** (each of the variables has zero mean and unit standard deviation).
- Otherwise the **clusters may be very stretched** and k-means will **perform poorly**.
- Scaling the features does not guarantee that all the clusters will be nice and spherical, but it **generally improves things**.

Let's see why standardization is important.

K-Means: Practical Issues

- A fundamental limitation of the Euclidean distance (or **Minkowski** distance in general) is that it does not take into account **how the data is distributed**.

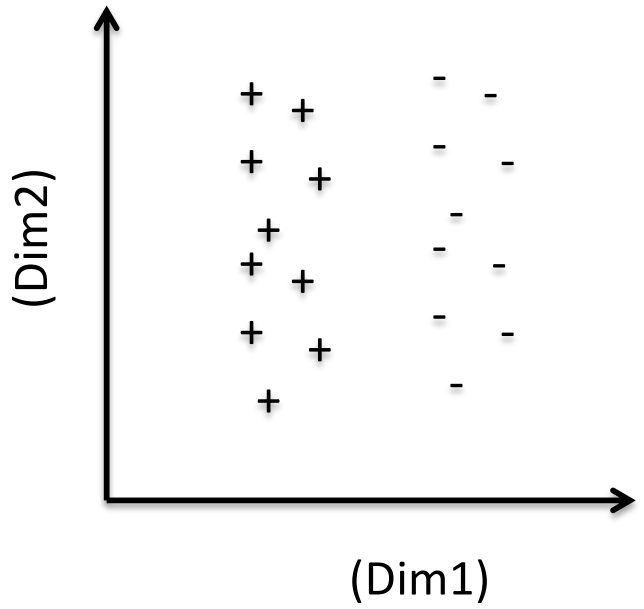


For example, along dimension 2 data has **large variance (spread)**.

So, distance between two points will be **overweighed along dim 2**.

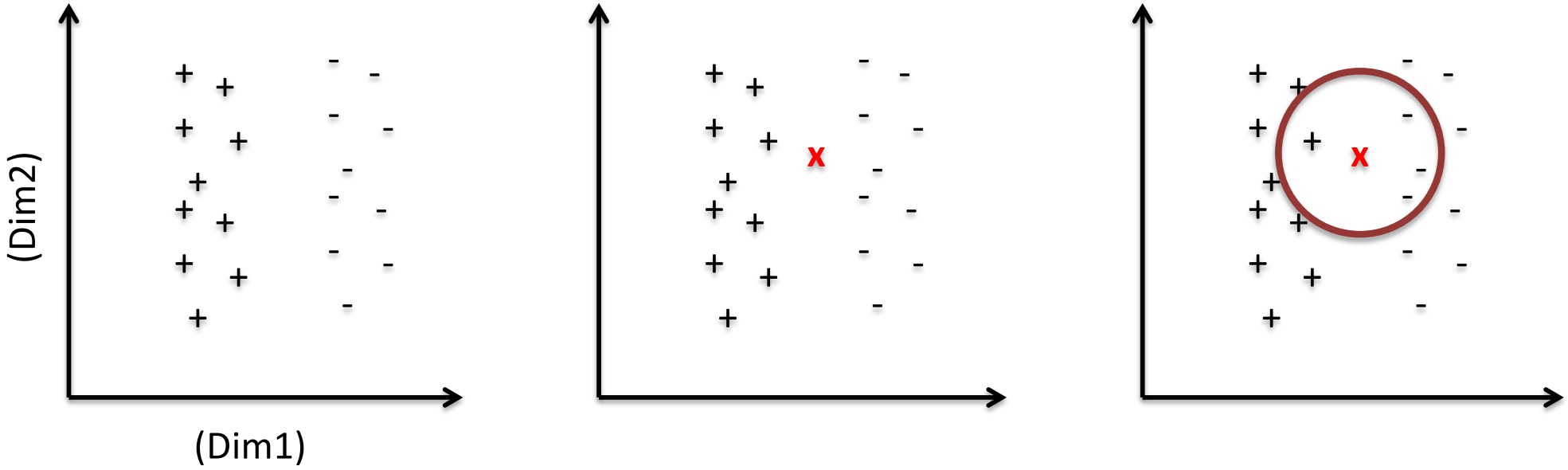
K-Means: Practical Issues

- In the figure, Dim1 could be in meter & Dim2 could be mm.
- So, there is a **large variation in the scale (unit) of the dimensions**.
- As a consequence Euclidean distance will **not provide the correct similarity measure**.



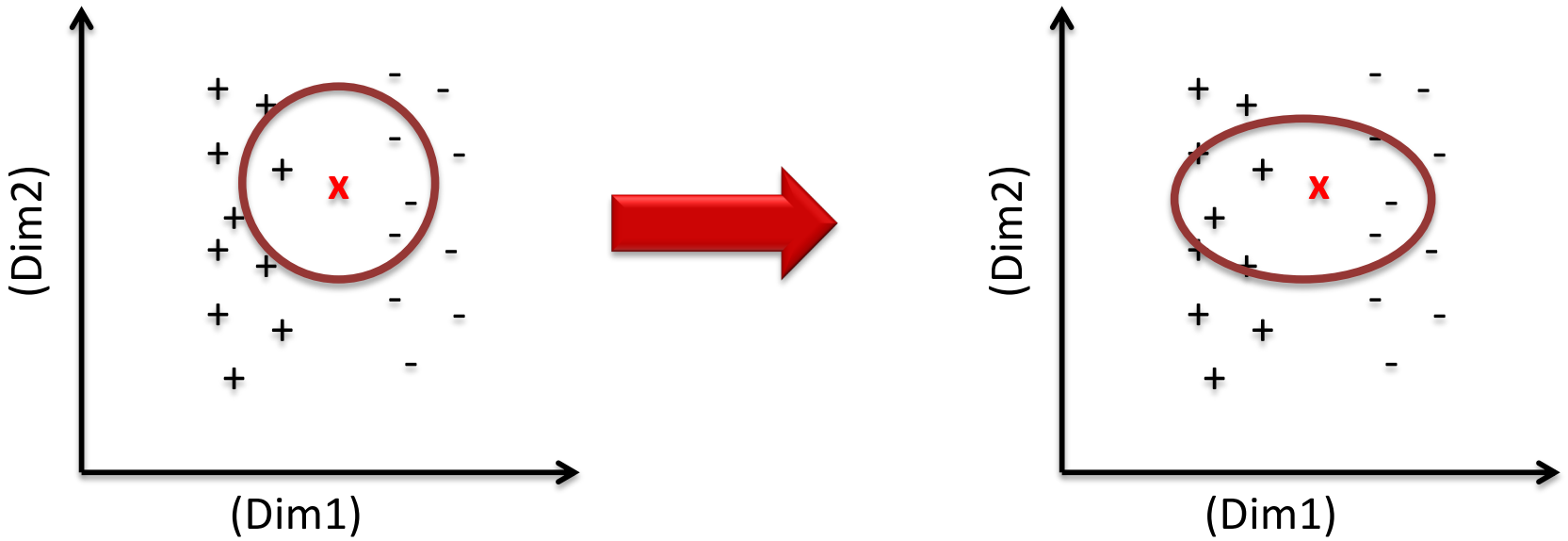
K-Means: Practical Issues

- Let's say that we want to find the closest samples of "**x**".
- Since the **dimensions are off**, due to Euclidean distance, some samples will be closer as compared to other "similar" points.
- So, we need to **adjust (rescale) the distance**.



K-Means: Practical Issues

- We can **rescale** the distance by **dividing the distance in each dimension by its variance**.
- We define, $d^2(\vec{x}, \vec{z}) = (x_1 - z_1)^2/\sigma_1^2 + (x_2 - z_2)^2/\sigma_2^2$
- This is how we **fix the scaling problem**.



K-Means: Practical Issues

- For an **arbitrary d-dimensional vector**, we can **generalize** the scaled distance measure as:

For every **single coordinate/dimension** we have a scaling factor σ

$$d^2(\vec{x},\vec{z}) = \sum_{i=1}^{d} \frac{(x_i - z_i)^2}{\sigma_i^2}$$

The variance matrix is ***diagonal***

$$d^2(\vec{x},\vec{z}) = (\vec{x}-\vec{z})^T \begin{bmatrix} \frac{1}{\sigma_1^2} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{\sigma_d^2} \end{bmatrix} (\vec{x}-\vec{z})$$

K-Means: Practical Issues

- Feature scaling is also known as **data standardization** and is generally performed during the **data preprocessing step**.
- First we make an assumption that the distribution of the features are **Gaussian**.

We determine the distribution **mean** and **standard deviation** for each feature.

$$x' = \frac{x - \bar{x}}{\sigma}$$

Next we **subtract the mean** from each feature.

Then we **divide the values** (mean is already subtracted) of each feature by its standard deviation.

This process standardizes features by **removing the mean** and **scaling to unit variance**

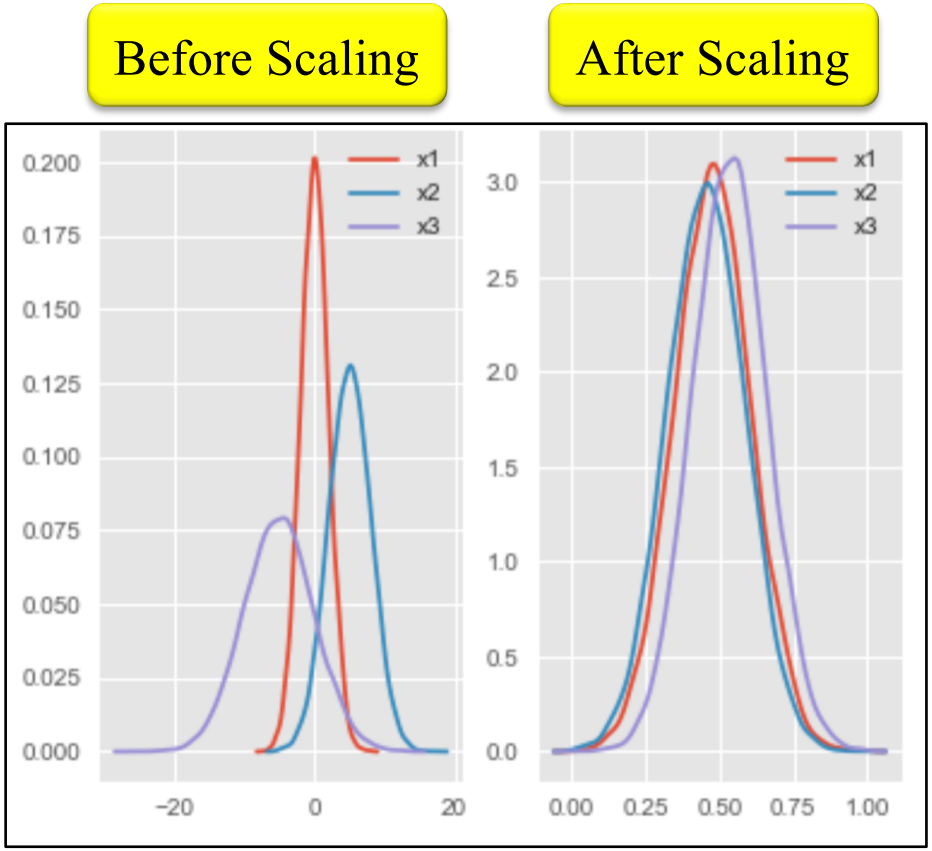
K-Means: Practical Issues

This process standardizes features by **removing the mean** and **scaling to unit variance**

- We **subtract the mean** to **normalize the data**, so that in addition to scaling it is also **centered around its mean**.

$$x' = \frac{x - \bar{x}}{\sigma}$$

If data is **not normally distributed** then we can't use this scaling technique.



K-Means: Practical Issues

- Convergence to the Global Minimum:
- Note that the distortion function **J** is a **non-convex** function.
- So coordinate descent on J is **not guaranteed to converge** to the global minimum.

$$J = \sum_{i=1}^{N} \sum_{j=1}^{k} a_{ij} \parallel \vec{x_i} - \vec{\mu_j} \parallel^2$$

K-Means: Practical Issues

- In other words, k-means can be **susceptible to local optima**.
- Very often k-means will work fine and come up with **very good clustering** ***despite this***.

$$J = \sum_{i=1}^{N} \sum_{j=1}^{k} a_{ij} \parallel \overrightarrow{x_i} - \overrightarrow{\mu_j} \parallel^2$$

K-Means: Practical Issues

- To *avoid* getting stuck in bad local minima, one common thing to do is **run k-means many times**.
- For this use **different random initial values** for the cluster centroids μ_j.
- Then, out of all the different clusterings found, pick the one that **gives the lowest distortion J**.

Another approach is to perform a **clever initialization** that guarantees lowest J.

Initialization & Avoiding Local Minima

Initialization and Avoiding Local Minima

- It is common to **pick initial k cluster centers at random**.
- However, we can choose these cluster centers in a clever fashion to **guarantee optimality**.
- It is given by the **k++ Means** algorithm.
- It was proposed in **2006** by David Arthur and Sergei Vassilvitskii.

Initialization and Avoiding Local Minima

- The k++ Means algorithm **picks the centers sequentially**.
- The goal is to "cover" the entire dataset.

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 $k \leftarrow 2 \ldots K$ **do** ▷ Loop over the rest of the centers.
  for $n \leftarrow 1 \ldots N$ **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 $n \leftarrow 1 \ldots N$ **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
  $n \leftarrow \text{Discrete}(p_1, p_2, \ldots, p_N)$ ▷ Draw a datum from this distribution.
  $\boldsymbol{\mu}_k \leftarrow \boldsymbol{x}_n$ ▷ Make this datum the next center.
end for
Return cluster means $\{\boldsymbol{\mu}_k\}_{k=1}^K$.

Initialization and Avoiding Local Minima

- We pick the **initial point** **uniformly at random**.
- Make this point the **first cluster center**.

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 $k \leftarrow 2 \ldots K$ **do** ▷ Loop over the rest of the centers.
  for $n \leftarrow 1 \ldots N$ **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 $n \leftarrow 1 \ldots N$ **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
  $n \leftarrow \text{Discrete}(p_1, p_2, \ldots, p_N)$ ▷ Draw a datum from this distribution.
  $\boldsymbol{\mu}_k \leftarrow \boldsymbol{x}_n$ ▷ Make this datum the next center.
end for
Return cluster means $\{\boldsymbol{\mu}_k\}_{k=1}^K$.

Initialization and Avoiding Local Minima

- Then **each subsequent point** is picked from the remaining points with **probability proportional to its squared distance to the point's closest cluster center**.

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 $k \leftarrow 2 \ldots K$ **do** ▷ Loop over the rest of the centers.
  for $n \leftarrow 1 \ldots N$ **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 $n \leftarrow 1 \ldots N$ **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
  $n \leftarrow \text{Discrete}(p_1, p_2, \ldots, p_N)$ ▷ Draw a datum from this distribution.
  $\boldsymbol{\mu}_k \leftarrow \boldsymbol{x}_n$ ▷ Make this datum the next center.
end for
Return cluster means $\{\boldsymbol{\mu}_k\}_{k=1}^K$.

Initialization and Avoiding Local Minima

- Surprisingly, this simple trick can be shown to *guarantee* that the distortion is never more than **O(log K)** worse than optimal.

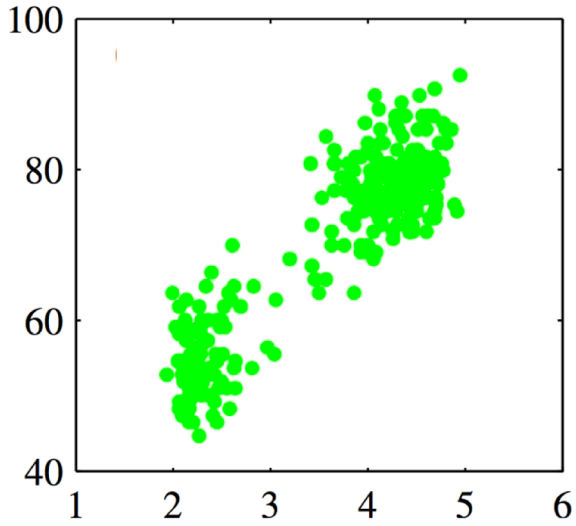
Input: Data vectors $\{\boldsymbol{x}_n\}_{n=1}^N$, number of clusters K
$n \leftarrow \text{RandomInteger}(1, N)$ ▷ Choose a datum at random.
$\boldsymbol{\mu}_1 \leftarrow \boldsymbol{x}_n$ ▷ Make this random datum the first cluster center.
for $k \leftarrow 2 \ldots K$ **do** ▷ Loop over the rest of the centers.
  for $n \leftarrow 1 \ldots N$ **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 $n \leftarrow 1 \ldots N$ **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
  $n \leftarrow \text{Discrete}(p_1, p_2, \ldots, p_N)$ ▷ Draw a datum from this distribution.
  $\boldsymbol{\mu}_k \leftarrow \boldsymbol{x}_n$ ▷ Make this datum the next center.
end for
Return cluster means $\{\boldsymbol{\mu}_k\}_{k=1}^K$.

K-Means Algorithm: Illustration

K-Means

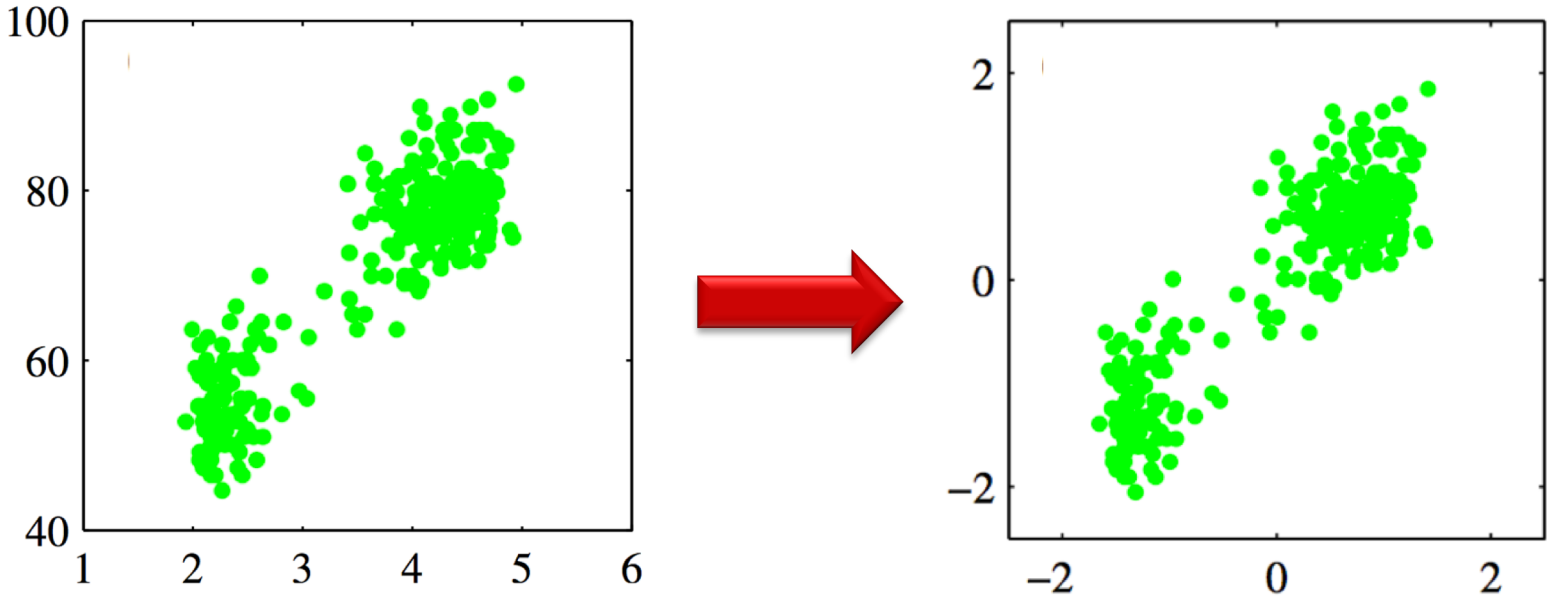
- The k-means algorithm is illustrated using the **Old Faithful data set**.
- It comprises **272 measurements of the eruption of the Old Faithful geyser** at Yellowstone National Park in the USA.
- Each measurement comprises the **duration of the eruption** in minutes (horizontal axis) and the time in minutes to the **next eruption** (vertical axis).

The data set forms **two dominant clumps**.



K-Means

- To apply the k-Means algorithm, the data needs to be **standardized** (each of the variables has **zero mean** and unit standard deviation).



K-Means

- For this example, we have chosen K = 2.
- So in this case, the assignment of **each data point to the nearest cluster center is equivalent** to a **classification** of the data points according to which side they lie of the perpendicular bisector of the two cluster centers.

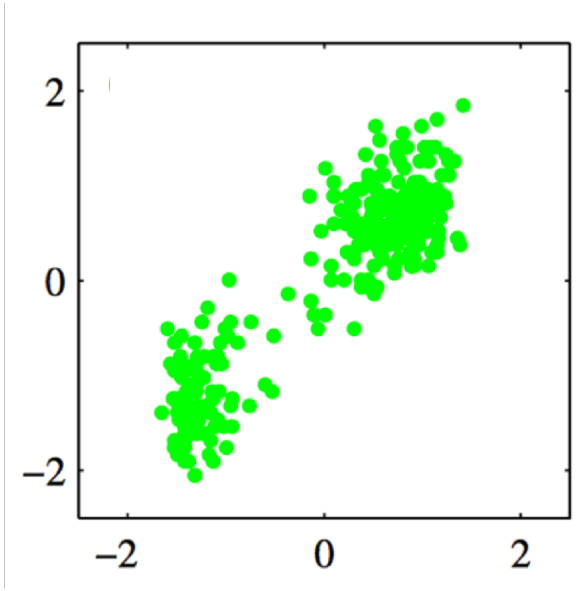
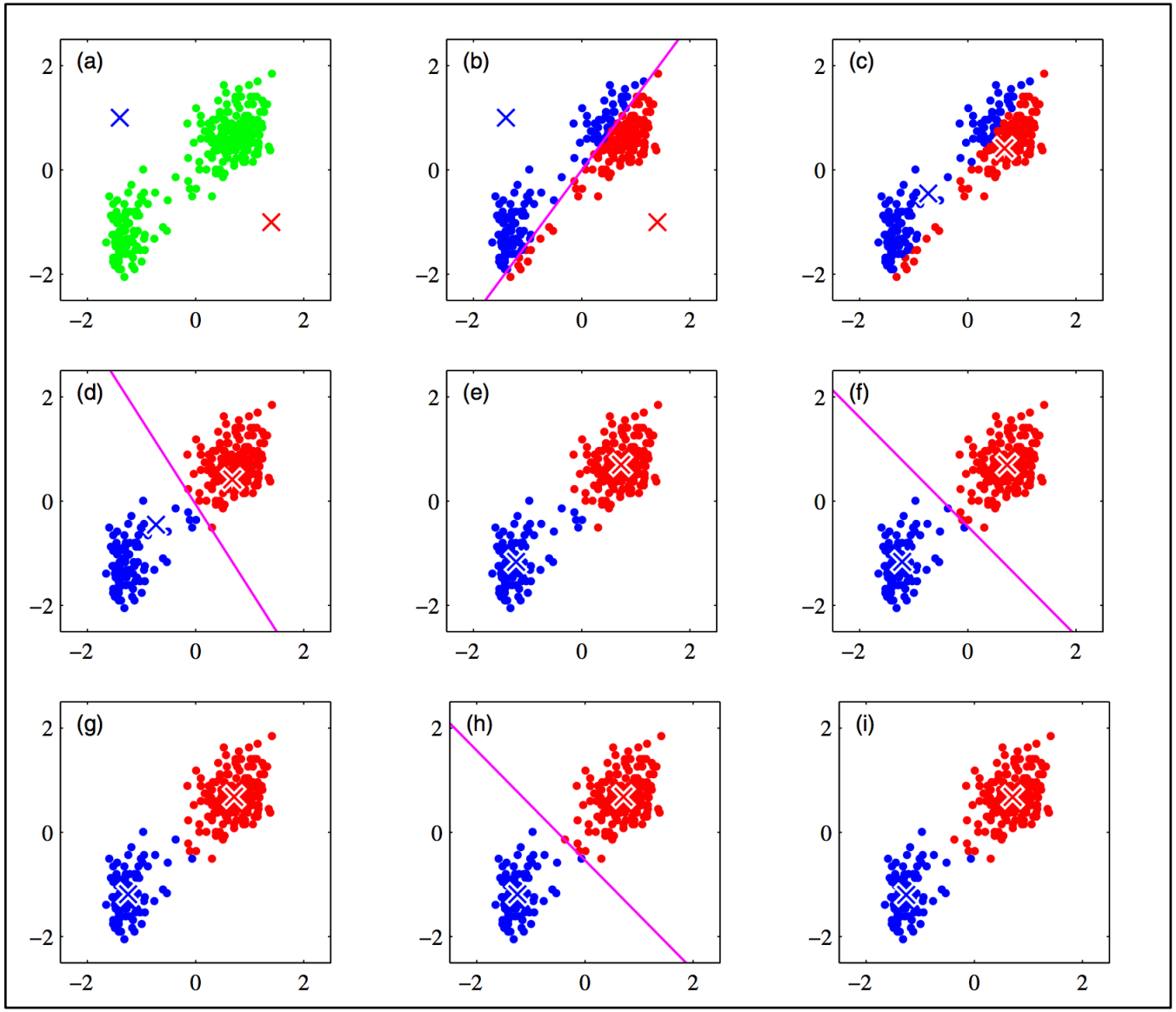


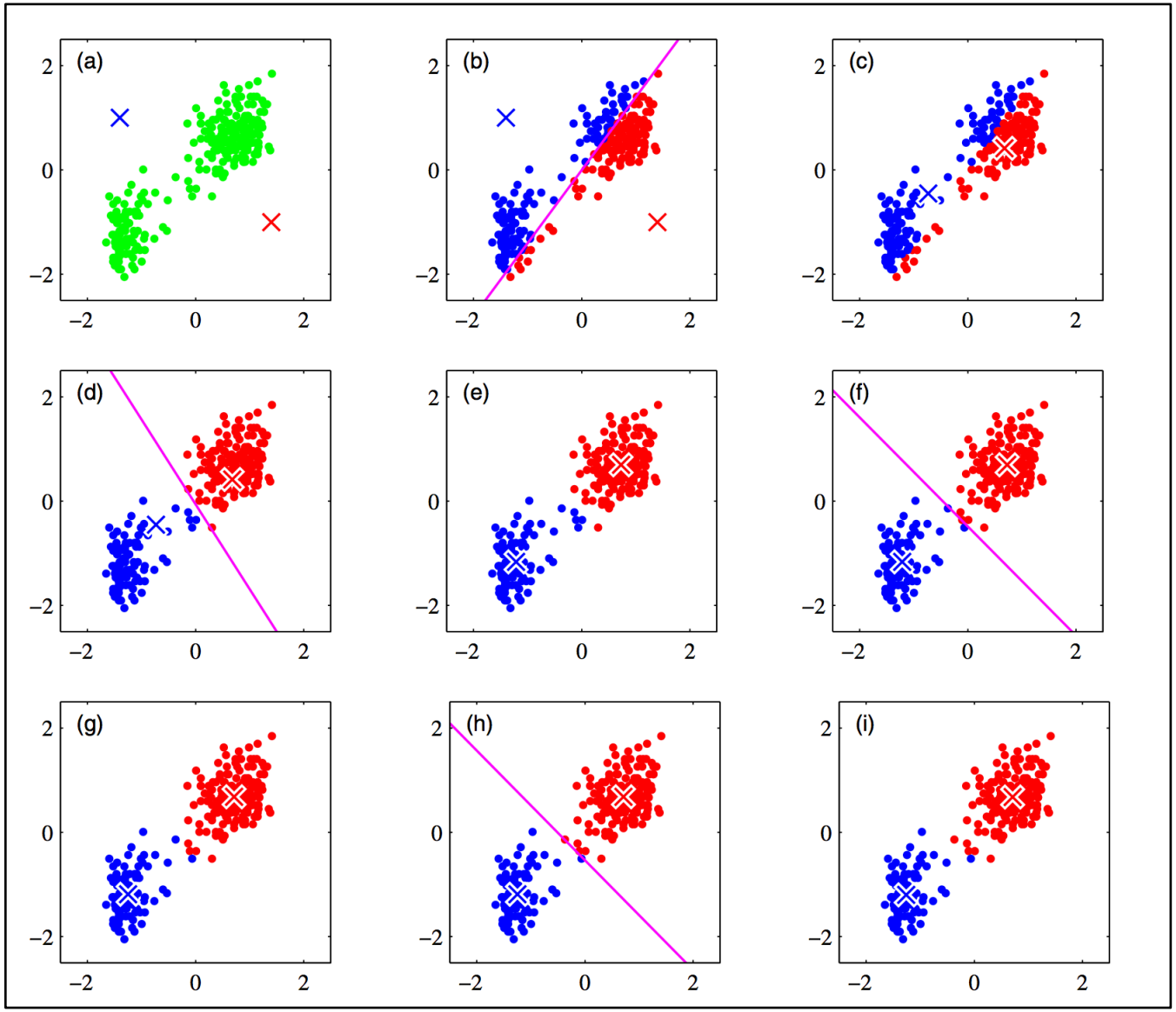
Illustration of the K-means algorithm using the re-scaled Old Faithful data set



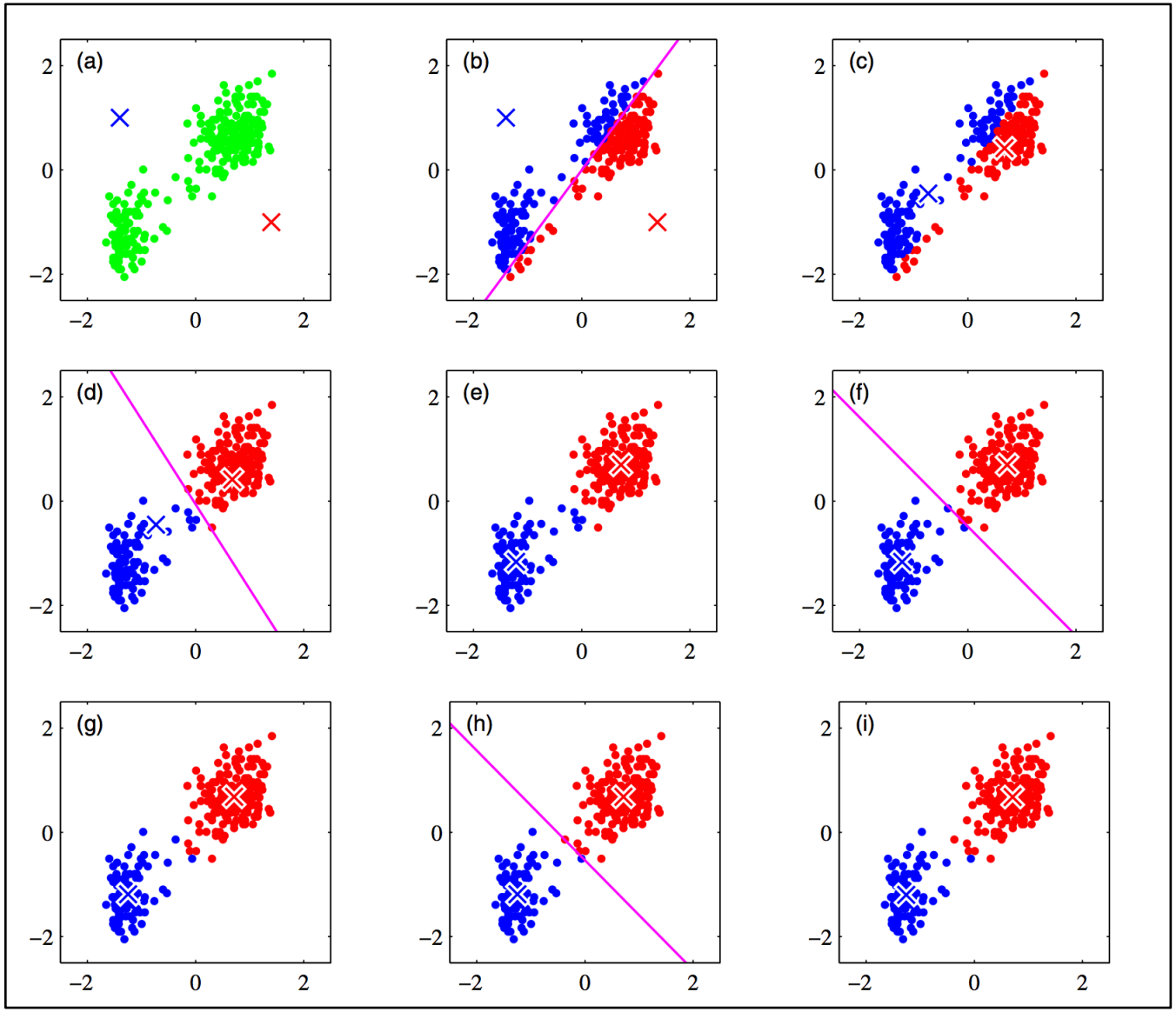
(a) Green points denote the data set in a 2D Euclidean space.

(a) The initial choices for centers μ_1 and μ_2 are shown by the red and blue crosses, respectively.

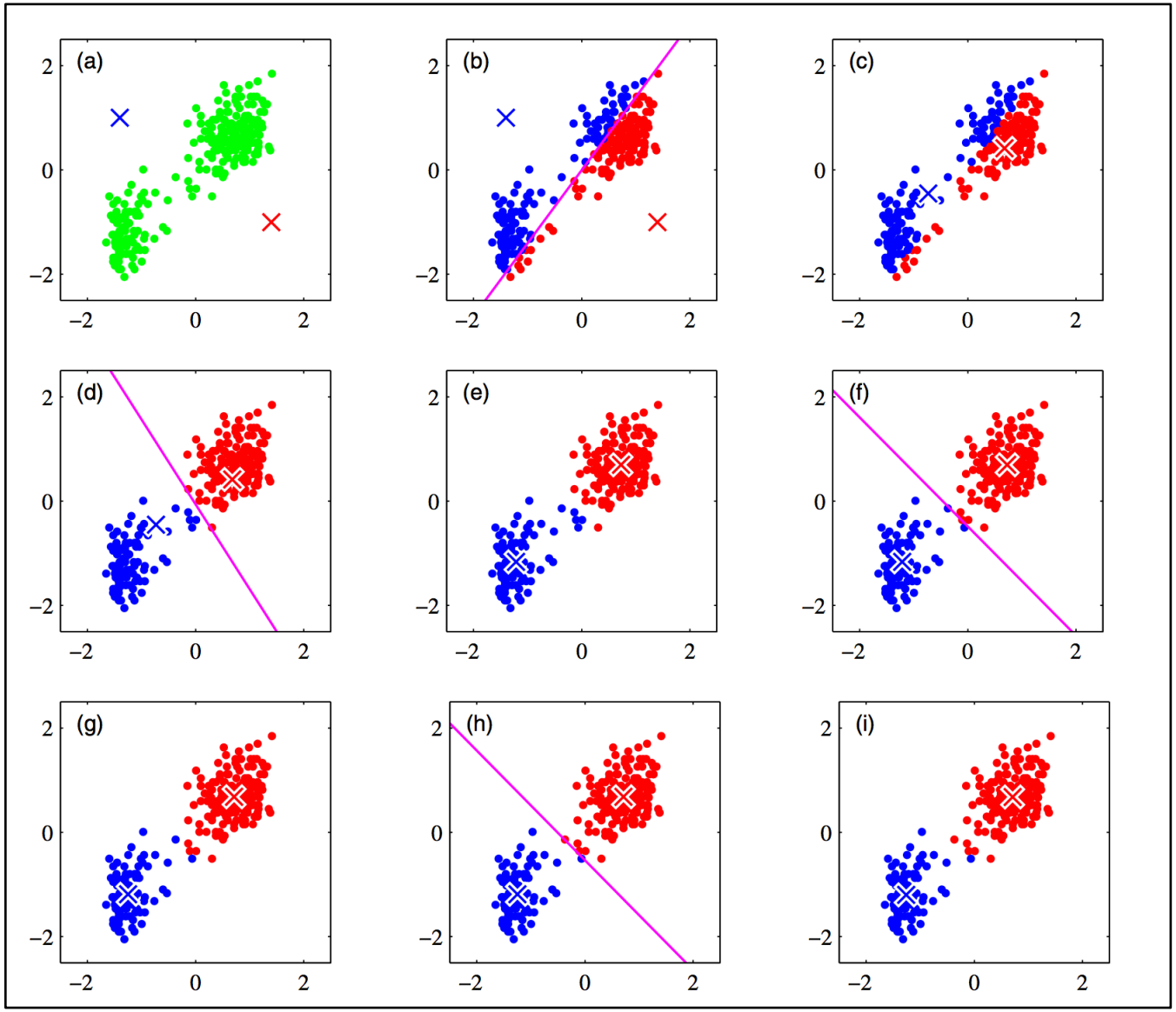
(b) In the **initial E step**, each data point is assigned either to the red cluster or to the blue cluster, according to **which cluster center is nearer**.



This is equivalent to classifying the points according to **which side of the perpendicular bisector** of the two cluster centers, shown by the **magenta line**, they lie on.



(c) In the subsequent **M step**, each **cluster center is re-computed** to be the **mean of the points** assigned to the corresponding cluster.

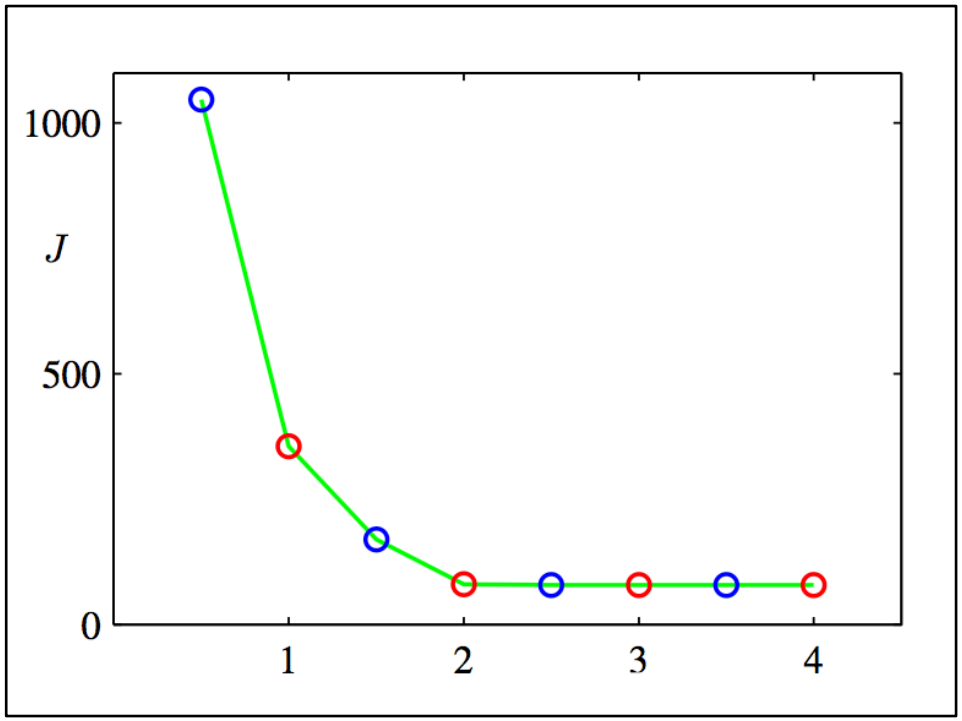


(d)–(i) show **successive E and M steps** through to final convergence of the algorithm.

K-Means

- A **plot of the cost function J** for the Old Faithful example.
- J is represented after each E step (blue points) and M step (red points) of the k-means algorithm.

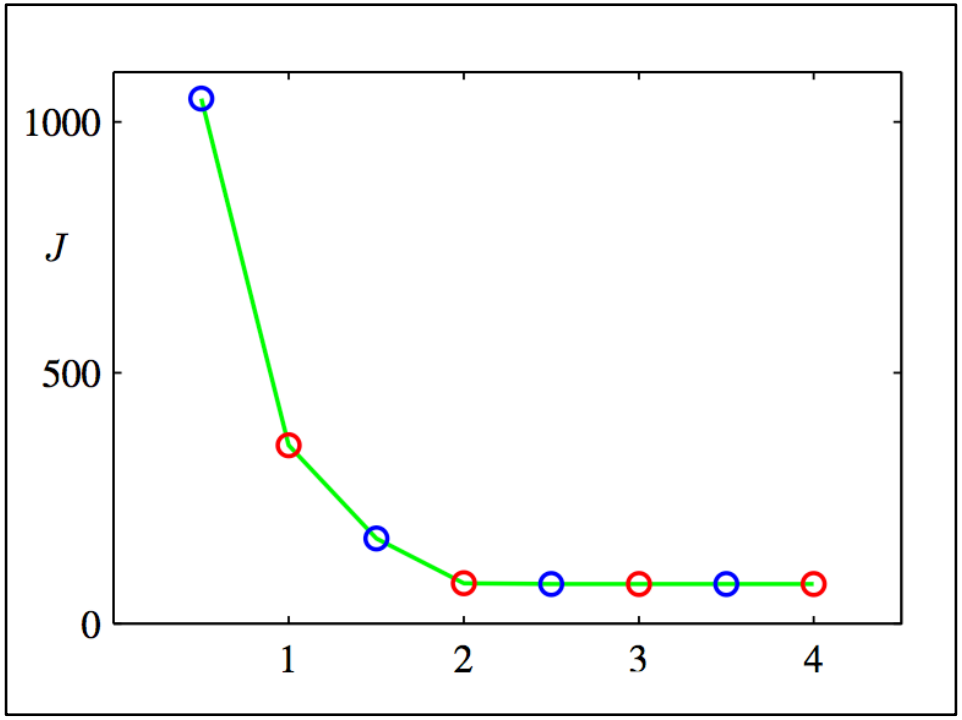
$$J = \sum_{i=1}^{N} \sum_{j=1}^{k} a_{ij} \parallel \vec{x_i} - \vec{\mu_j} \parallel^2$$



K-Means

- The algorithm has **converged after the 3rd M step**.
- The **final EM cycle produces no changes** in either the assignments or the prototype vectors.

$$J = \sum_{i=1}^{N} \sum_{j=1}^{k} a_{ij} \parallel \vec{x_i} - \vec{\mu_j} \parallel^2$$

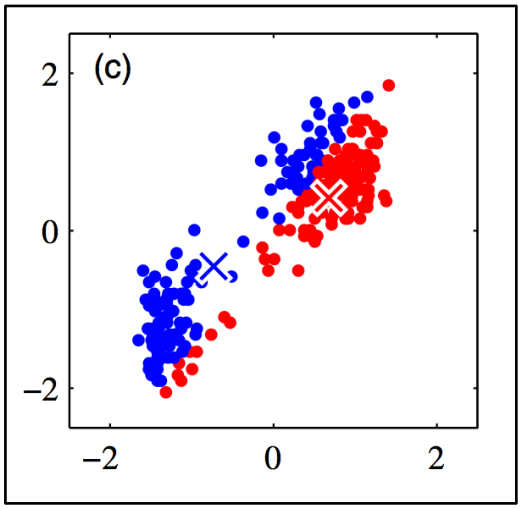


K-Means: Computational Complexity

K-Means: Complexity

- A **direct implementation** of the K-means algorithm can be **relatively slow**.
- Because in each E step it is necessary to compute the **Euclidean distance between N data points** (d dimensional) and **K cluster centers**.
- It takes **O(NKd)** time.

$$J = \sum_{i=1}^{N} \sum_{j=1}^{k} a_{ij} \parallel \vec{x_i} - \vec{\mu_j} \parallel^2$$



K-Means: Complexity

- Various schemes have been proposed for **speeding up** the K-means algorithm.
- Based on **precomputing a data structure** such as a tree, such that nearby points are in the same subtree.
- Other approaches make use of the **triangle inequality for distances**, thereby avoiding unnecessary distance calculations.

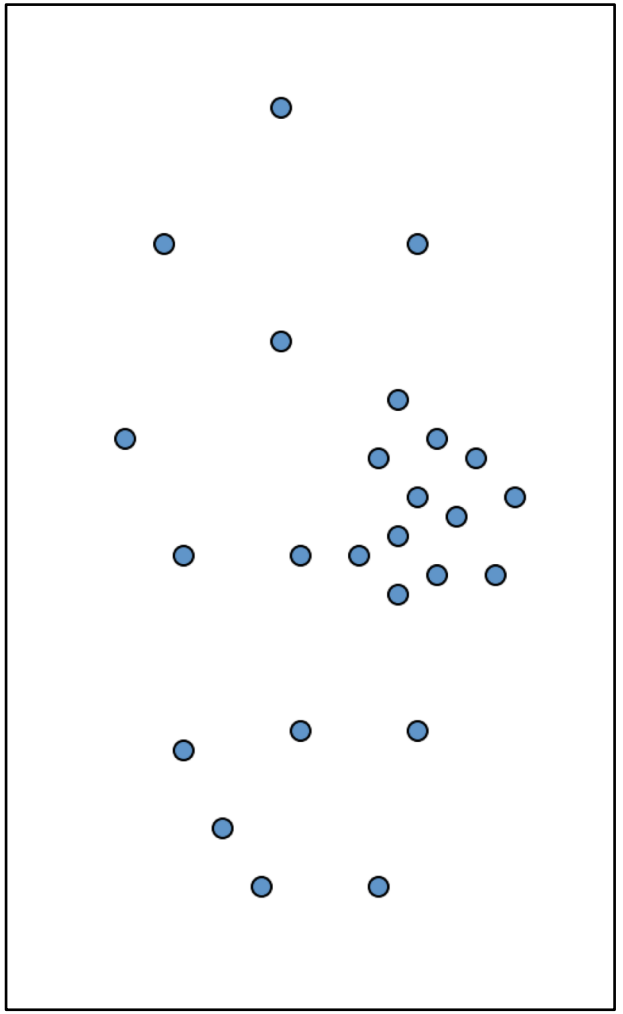
K-Means Algorithm: Limitation

K-Means: Limitation

- To run the k-Means algorithm we have **to know the value of "k"**.
- It requires some **domain knowledge**.
- Choosing an optimal k is **non-trivial**.
- See the following notebook for an empirical treatment of this issue:
- https://github.com/rhasanbd/Clustering-K-Means-All-You-Care-About/blob/master/Clustering-2-K-Means-How%20to%20Choose%20Optimal%20K.ipynb

K-Means: Limitation

- To understand another limitation, consider the data distribution in the figure.
- K-means **doesn't work well** on this type of data.
- Because **clusters may overlap**.
- Some clusters may be "wider" or oblong than other clusters.



This limitation is solved by using a **Gaussian Mixture Model (GMM)**.

K-Means: Empirical Understanding

- For an empirical understanding of the K-Means clustering algorithm, see the notebook "***Clustering- K-Means: All You Care About***":
- https://github.com/rhasanbd/Clustering-K-Means-All-You-Care-About

K-Means: Applications

K-Means: Applications

- K-means algorithm can be used to solve the following problems:

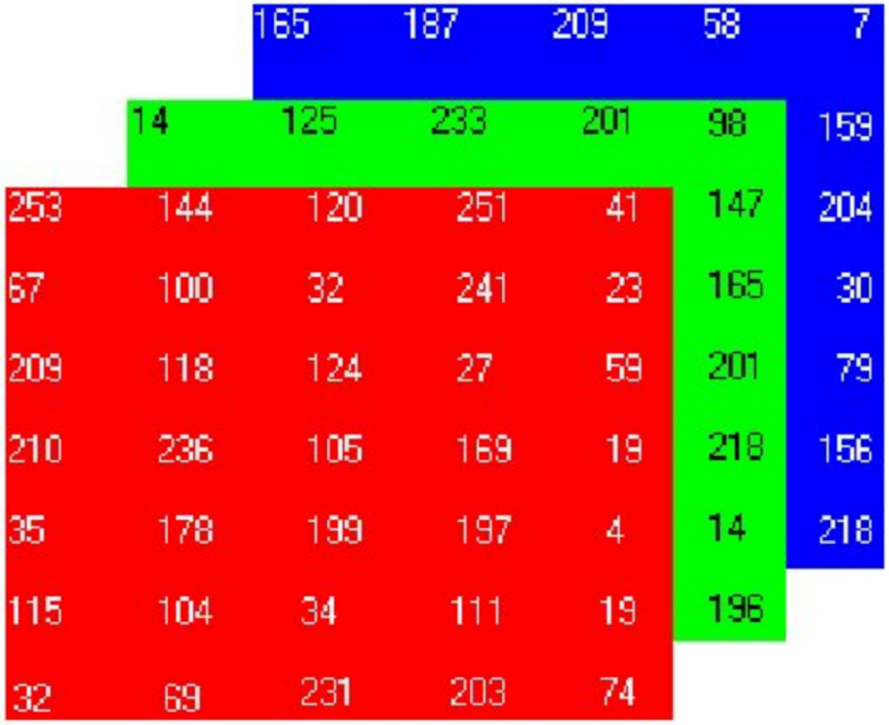
  - Image segmentation
  - Image compression

K-Means: Applications

- Image segmentation
- The goal of segmentation is to **partition an image** into regions.
- Each region should have a **reasonably homogeneous visual appearance** or correspond to objects or parts of objects.

K-Means: Applications

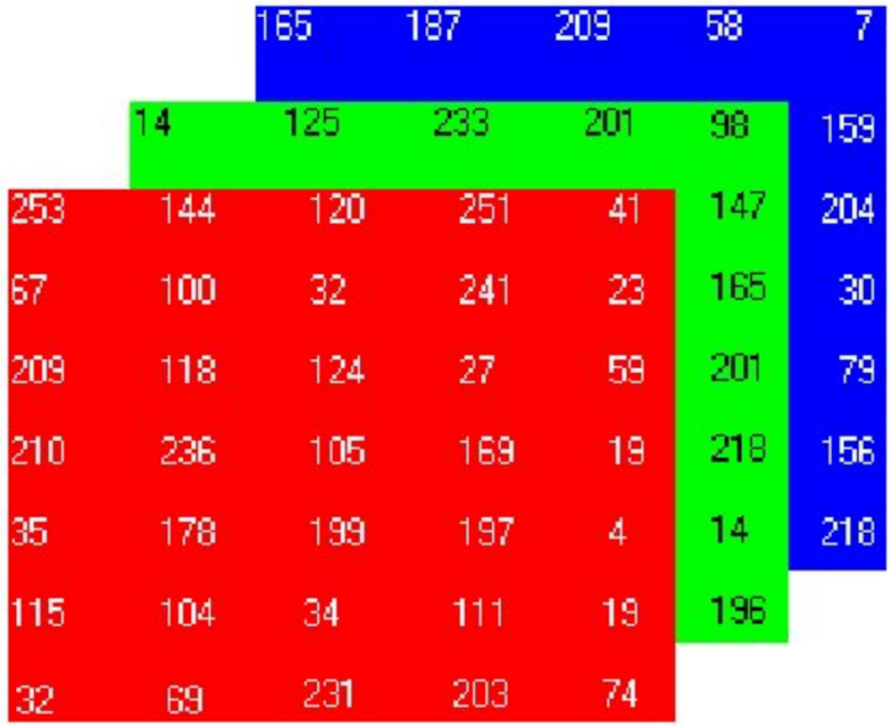
- Each **pixel** in an image is a point in a **3-dimensional space** comprising the intensities of the red, blue, and green channels.
- It's a *m x m x 3* array, where the 3rd dimension is the number of color channels.



K-Means: Applications

- The segmentation algorithm simply **treats each pixel in the image as a separate data point**.
- For example, if we reshape the m x m x 3 array to m^2 x 3 array, then each row represents a pixel.

Note that strictly this space is **not Euclidean** because the channel intensities are bounded by the interval [0, 1].



K-Means: Applications

- Nevertheless, we can apply the K-means algorithm without difficulty.
- We illustrate the result of running K-means to convergence, for **any particular value of k**.
- This is done by re-drawing the image replacing each pixel vector with the {R, G, B} intensity triplet given by the center μ_j to which that pixel has been assigned.
- Results for **various values of k** are shown in the following figure.

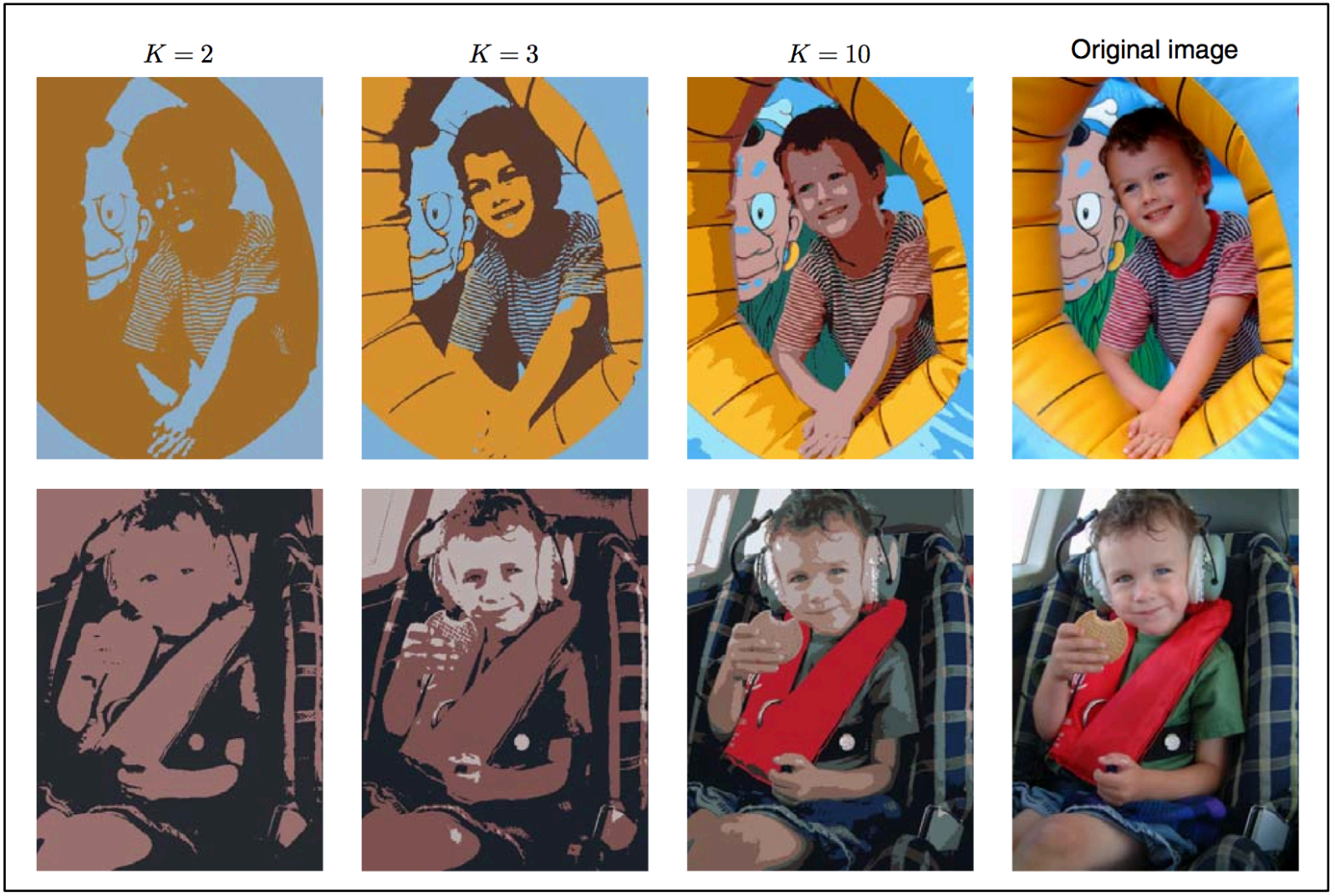


Image segmentation by the k-means clustering algorithm: the initial images together with their **K-means segmentations** obtained using various values of k.

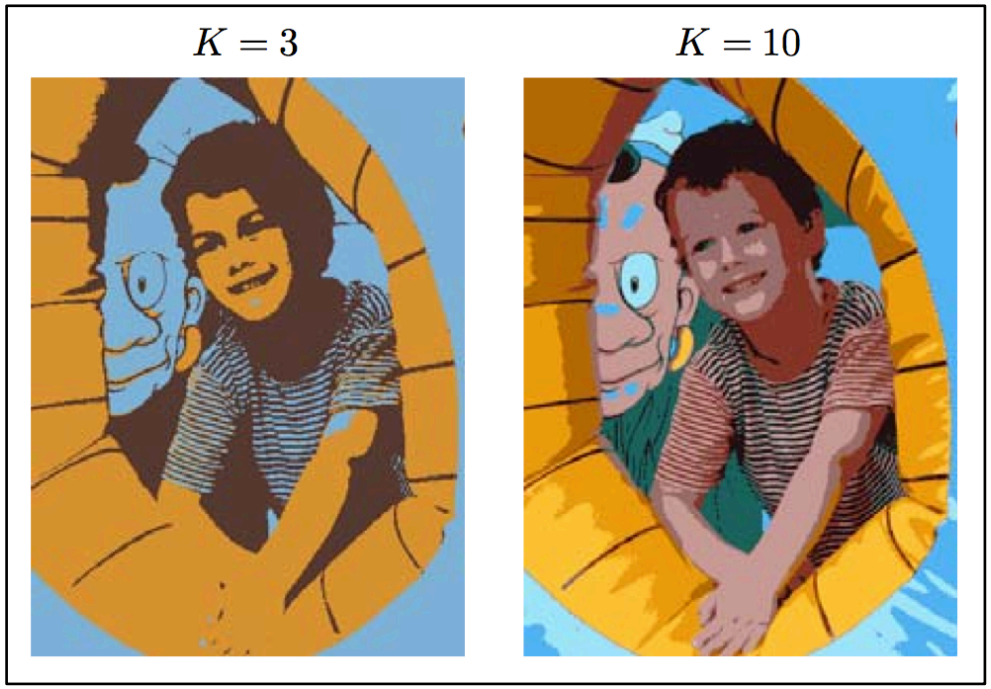
K-Means: Applications

- We see that for a given value of k, the algorithm is representing the image using a **palette of only k colors**.
- It should be emphasized that this use of k-means is not a particularly **sophisticated approach to image segmentation**.

Because it takes no account of the **spatial proximity of different pixels**.

K-Means: Applications

- The image segmentation problem is in **general extremely difficult** and remains the subject of active research and is introduced here simply to illustrate the behavior of the k-means algorithm.



K-Means: Applications

- We can also use the result of a clustering algorithm to perform **data compression**.
- It is important to **distinguish between** lossless and lossy data compression.

K-Means: Applications

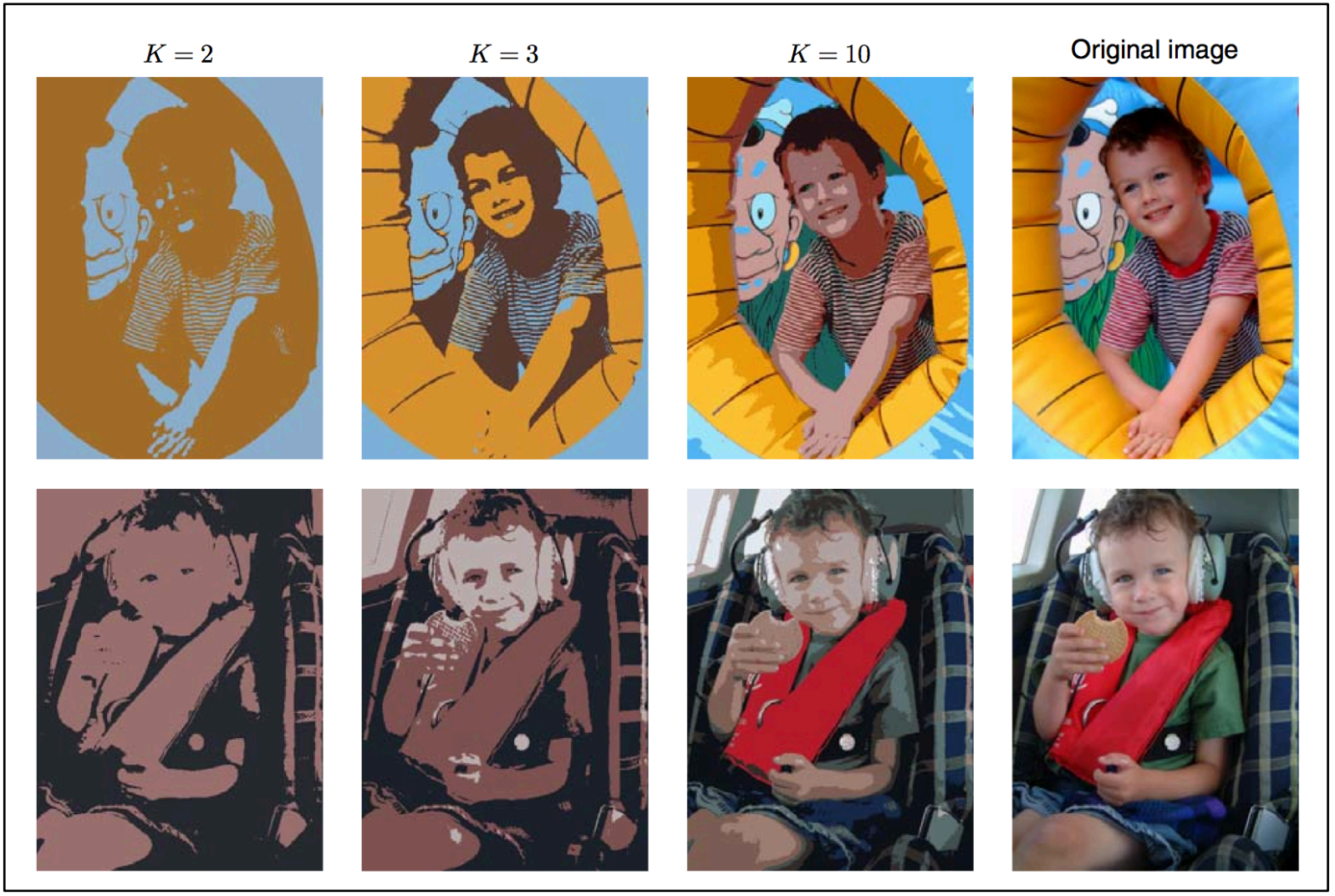
- **Lossless compression**: the goal is to be able to reconstruct the original data **exactly** from the compressed representation.
- **Lossy compression**: we ***accept some errors*** in the reconstruction in return for higher levels of compression than can be achieved in the lossless case.

K-Means: Applications

- We can apply the k-means algorithm to the problem of **lossy data compression** as follows.
- For each of the N data points, we **store only the identity k of the cluster** to which it is assigned.
- We also store the values of the **k cluster centers μ_j**,
- It typically requires **significantly less data**, provided we choose $k \ll N$.

K-Means: Applications

- Each data point is then **approximated by its nearest center μ_j**.
- **New data points can similarly be compressed** by first finding the nearest μ_j.
- Then **storing the label k** instead of the original data vector.
- This framework is often called **vector quantization**, and the vectors μ_j are called **code-book vectors**.



Vector quantization for data compression: **smaller values of k** give **higher compression** at the expense of poorer image quality.

K-Means: Applications

- The image segmentation problem discussed also provides an illustration of the **use of clustering for data compression**.
- Suppose the original image has **N pixels** comprising {R, G, B} values each of which is stored with **8 bits of precision**.
- Then to transmit the whole image directly would cost **24N bits**.

K-Means: Applications

- Now suppose we first run k-means on the image data, and then instead of transmitting the original pixel intensity vectors we **transmit the identity of the nearest vector μ_j**.
- Because there are k such vectors, this requires **$\log_2 k$ bits per pixel**.

K-Means: Applications

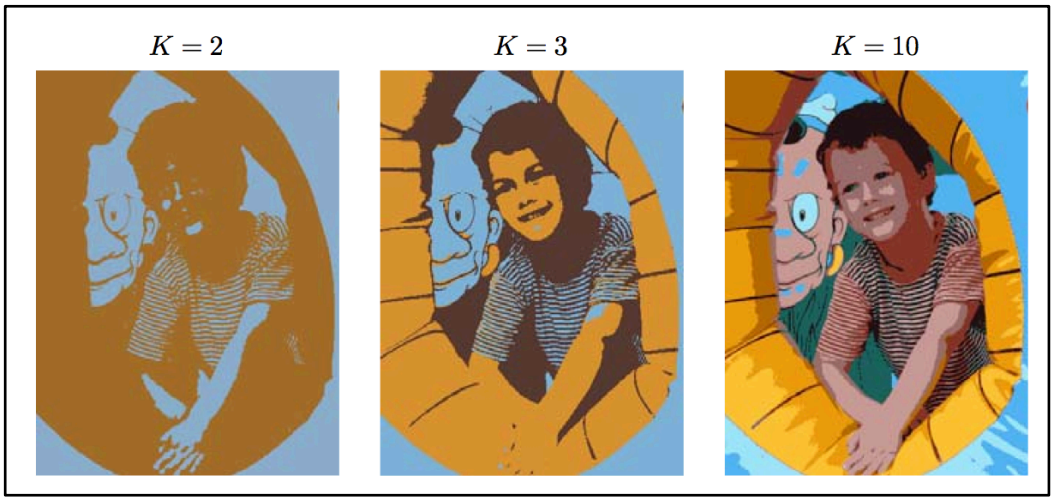
- We must also **transmit the k code book vectors μ_j**, which requires 24k bits.
- So the **total number of bits** required to transmit the image is **$24k + N \log_2 k$** (rounding up to the nearest integer).

K-Means: Applications

- The original image in the Figure has 240 × 180 = **43, 200 pixels**.
- So, it requires 24 × 43, 200 = **1, 036, 800 bits** to transmit directly.

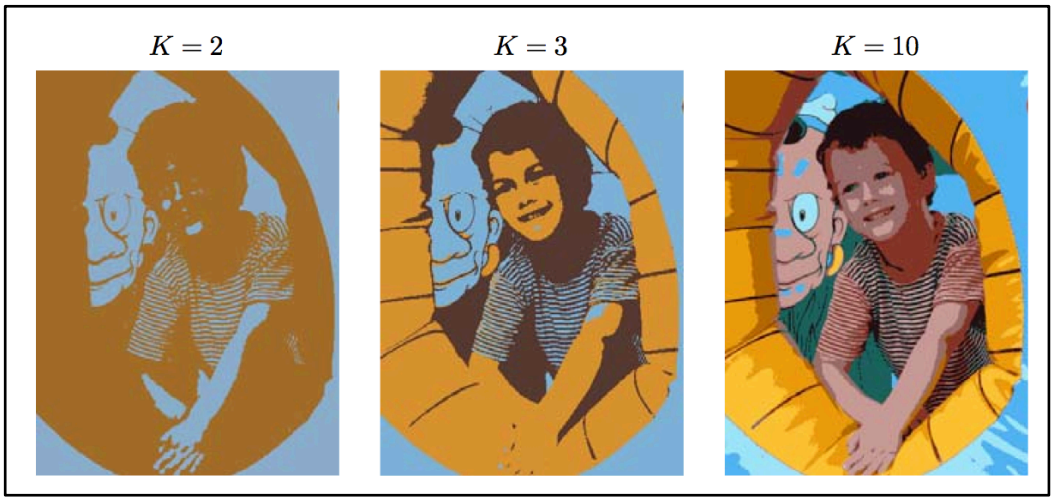
K-Means: Applications

- By comparison, the compressed images require **43, 248 bits** (K = 2), **86, 472 bits** (K = 3), and **173, 040 bits** (K = 10), respectively, to transmit.
- These represent **compression ratios** compared to the original image of 4.2%, 8.3%, and 16.7%, respectively.



K-Means: Applications

- We see that there is a **trade-off** between degree of compression and image quality.
- Note that our aim in this example is to **illustrate the k-means algorithm**.



K-Means: Applications

- If we had been aiming to produce a **good image compressor**, then it would be more fruitful to **consider small blocks of adjacent pixels**.
- For instance 5 × 5, and thereby **exploit the correlations** that exist in natural images between nearby pixels.